

ESD-TR-70-377



ANALYSIS OF MAJOR COMPUTER OPERATING SYSTEMS

August 1970

DEPUTY FOR COMMAND AND MANAGEMENT SYSTEMS
HQ ELECTRONIC SYSTEMS DIVISION (AFSC)
L. G. Hanscom Field, Bedford, Massachusetts 01730

This document has been
approved for public release and
sale; its distribution is
unlimited.

ESTI FILE COPY

(Prepared under Contract No. F19628-70-C-0258 by The COMTRE Corporation,
151 Sevilla Avenue, Coral Gables, Florida 33134.)

AD715919

LEGAL NOTICE

When U. S. Government drawings, specifications or other data are used for any purpose other than a definitely related government procurement operation, the government thereby incurs no responsibility nor any obligation whatsoever; and the fact that the government may have formulated, furnished, or in any way supplied the said drawings, specifications, or other data is not to be regarded by implication or otherwise as in any manner licensing the holder or any other person or conveying any rights or permission to manufacture, use, or sell any patented invention that may in any way be related thereto.

OTHER NOTICES

Do not return this copy. Retain or destroy.

ANALYSIS OF MAJOR COMPUTER OPERATING SYSTEMS

August 1970

DEPUTY FOR COMMAND AND MANAGEMENT SYSTEMS
HQ ELECTRONIC SYSTEMS DIVISION (AFSC)
L. G. Hanscom Field, Bedford, Massachusetts 01730

This document has been
approved for public release and
sale; its distribution is
unlimited.

(Prepared under Contract No. FI9628-70-C-0258 by The COMTRE Corporation,
151 Sevilla Avenue, Coral Gables, Florida 33134.)




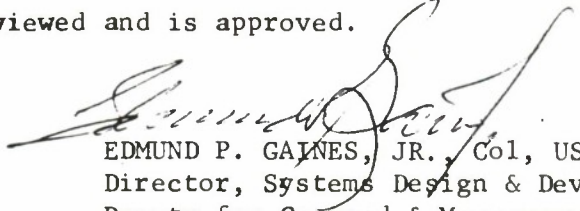
FOREWORD

This report presents the results of an analysis conducted by The COMTRE Corporation of Coral Gables, Florida in support of Project 6917, Task 691701 under Contract Number F19628-70-C-0258. This work is a continuation of an earlier effort documented in ESD-TR-70-64 (AD 704 138), 31 January 1970. The earlier report describes the investigation of the executive/control capabilities of contemporary computer operating systems. This report expands that investigation to include the supporting software capabilities of vendor-supplied operating systems. Thus, this report establishes a more comprehensive baseline for further analyses.

The analysis recorded in this report was performed by Messrs. Clinton S. McIntosh, Kenneth P. Choate and William C. Mittwede. Dr. John B. Goodenough (ESD/MCDS) was the ESD Project Monitor.

This technical report has been reviewed and is approved.


ALFRED J. BEAUCHAMP
Technical Director
Directorate of Systems Design and
Development
Deputy for Command & Management Systems


EDMUND P. GAINES, JR., Col, USAF
Director, Systems Design & Development
Deputy for Command & Management Systems

ABSTRACT

This report presents the results of an analysis of approximately 50 contemporary computer operating systems. The operating systems were analyzed for the purpose of defining an integrated functional classification structure applicable and common to the executive/control functions, system management functions and data manipulation functions of commercial operating systems. In addition to the functional classification structure, a tutorial discussion is presented to describe functional capabilities of contemporary operating systems in terms of common identified functions, and selected implementation variations that occur within real-time, time-sharing, and batch processing environments. A glossary of operating system terminology is also included.

CONTENTS

Section I	- Introduction	1
	1.1 Purpose	1
	1.2 Scope	1
	1.3 Organization of This Report	4
Section II	- Overview of Contemporary Operating Systems	5
	2.1 Evolution of Operating System Features	5
	2.2 Types of Third-Generation Operating Systems	7
	2.3 Operating System Levels	12
	2.4 Operating System Functions	15
Section III	- Discussion of Operating System Functions	34
	3.1 Introduction	34
	3.2 Discussion	34
	Part I: Executive/Control Functions	35
	Part II: System Management Functions	55
	Part III: Data Manipulation Functions	62
Appendix I	- Glossary of Operating System Terminology	73
Appendix II	- Bibliography of Operating Systems Documentation	115

SECTION I

INTRODUCTION

1.1 Purpose

This report is the first of a series currently being produced by The COMTRE Corporation for the Electronic Systems Division of the Air Force Systems Command. The second report in this series will provide a framework for translating operational data processing requirements into specific operating system selection criteria. The third will establish validation requirements for major computer operating systems.

To be effective, these reports must be based upon a coherent, comprehensive view of contemporary operating systems which emphasizes the commonalities of these systems rather than their differences. The purpose of this report is to provide such a view.

Generally, this view consists of regarding operating systems as collections of hierarchically related functions. At the higher levels of this hierarchy are functions common to all modern operating systems. At the lower levels, a particular function may, or may not, be provided by a given operating system. However, at these lower levels, functions can accurately be considered implementation techniques of higher levels. Thus, the hierarchical functional classification structure provides a common reference framework from which to consider any operating system.

Since it is comprehensive and generally applicable to contemporary operating systems, the functional classification scheme satisfies the needs of the other reports in this series. Additionally, it should be useful for other analyses or as an introduction to operating system concepts.

1.2 Scope

The scope of this report, and the functional classification scheme, is confined to software which is distributed as a part of the computer operating system package by the various operating system vendors. In addition to the operating system software which performs the "executive" or "control" functions for the computer, the operating system features performing system generation and maintenance, library and directory maintenance, data management, sorting and merging, and general utility functions are also included. Specifically excluded, however, are language compilers, assemblers and report generators except to the extent that the operating system provides special interface functions for these elements.

The functional classification structure was developed through an analysis of major operating systems, based upon documentation provided by system vendors. The following guidelines were used to limit the number of systems analyzed to a manageable, but representative, group:

- a. General-Purpose Orientation - The software must be of a general enough nature to satisfy a number of users with differing problem orientations. Special-purpose software, though usually quite sophisticated, allows the functional requirements of the specific application to dictate the form and content of the operating system. Thus, the functional characteristics of the operating system become derivatives of the functional requirements of the application rather than general operating system features.
- b. Third-Generation Development - The software should be recognizable as a third-generation system. Several advantages are obtained from this orientation. The most obvious, of course, is that it focuses attention on the current state of the art. While the number of second-generation systems is quite large, the continued evolvement of these systems has produced a proven nucleus basic to third-generation development. Thus, the better functional characteristics of the earlier systems are present in the newer systems. Where certain characteristics have not been carried forward, either new technology has resulted in new approaches or the function has been proven unnecessary. In either case, the ability of third-generation systems to handle second-generation problems indicates that second-generation capabilities have been preserved.
- c. Commercially Available - The system must be available for general procurement to preclude examination of those systems which are called general purpose but are actually tailored to very specific operating environments.
- d. Documentation Accessibility - The system must be in a sufficiently advanced state of development to have produced documentation that accurately describes the system organization attributes and functional interactions.
- e. Non-Subordinate System - The system must not be a subordinate subset of a more inclusive higher-level operating system. The functional analysis of the larger system would include the functional characteristics of all subordinate subsystems. Consequently, an analysis of the subordinate system would have been either repetitive or functionally incomplete.

All of the operating systems represented in the analysis are listed in Table 2.2 on pages 12 and 13. It should be pointed out that while there are some commercially available

operating systems which have been excluded from the analysis, the selection is quite comprehensive in terms of computer system size, orientation, and manufacturer.

The guidelines specifically excluded several prototype or proprietary systems from the analysis. For example, the MULTICS operating system, the T.H.E. multiprogramming system, the CP-67/CMS, and the multitude of specially-developed operating systems for time-sharing service bureaus have not been investigated. While it is probably quite true that many of these systems would significantly have altered the lines along which the functional classification scheme was developed, the purpose of the project was not to develop a scheme applicable to every conceivable operating system. Instead, the focus was specifically restricted to existing commercially available systems since these are the systems for which functionally-oriented selection, evaluation and validation criteria are currently required.

As a result, this classification scheme should not be viewed as conflicting with other schemes which attempt to either describe different system environments or that are used for other purposes. For example, Harry Katzan, Jr., in a report presented at the 1970 Spring Joint Computer Conference entitled "Operating Systems Architecture," describes five operating system types: multiprogramming, hypervisor multiprogramming, time-sharing, virtual systems, and tri-level operating systems. This classification scheme was developed to encompass a number of experimental and research-oriented systems, including some of those cited above. Consequently, the classification structure does not purport to be an inclusive representation of commercially available software. Nevertheless, since several of these system types are not represented by any commercially available system, this categorization can only be superficially applied to the commercial environment.

Similarly, many university courses in computer science postulate a series of functions that an operating system must perform, and then proceed to build a series of design philosophies extending from this base. While an elegant device for illustrating design concepts, these schemes are generally inappropriate for the selection, evaluation, and validation of operating systems. Since they are developed to illustrate concepts, they tend to describe the system as the designer, rather than the user, visualizes it. However, in system selection, it is the user's operational needs that must be addressed, not the design technique employed nor the relative sophistication of the system.

Therefore, this operating system functional classification scheme has been developed to reflect a user's view of the system rather than an internal view. Each major function is composed of features which are observable or measurable within an evaluation and validation framework. As a result, this approach, while not precluding other uses for the functional classification scheme, assures that the original objectives for developing the categorization are addressed.

1.3 Organization of This Report

The next Section of this report describes some of the general characteristics of operating systems, categorizes operating system types, and introduces the functional classification scheme.

Section 3 is a tutorial discussion which illustrates and describes in detail the various functional characteristics of operating systems within the format provided by the functional classification scheme.

Appendix I presents a glossary of operating system terminology and an addendum discussing terms with uncommon or conflicting definitions. Appendix II is a bibliography of the operating system publications upon which this analysis was based.

SECTION II

OVERVIEW OF CONTEMPORARY OPERATING SYSTEMS

2.1 Evolution of Operating System Features

Paralleling the evolution of general-purpose computer hardware systems from the early 1950's has been the evolution of the software programs which serve as the interface between the application programmer and the hardware. These support or interface programs form the nucleus of manufacturer-supplied software which has come to be called the operating system. And just as computers are said to have three generations of design, operating systems are also described in terms of these generations. The following paragraphs briefly describe each stage in the growth of operating system design technology.

First-generation "operating systems" (though they were not so designated at the time) consisted of a minimal number of manufacturer-supplied routines of general enough utility to merit distribution to most computer customers. Initially, these routines performed such functions as loading programs and displaying and dumping computer main storage. Many of these computer systems did not have hardware instructions for certain arithmetic operations such as multiplication and division, and as a result, subroutines to perform these operations were also distributed. Frequently, computer users replaced or rewrote these programs to meet their own needs. Eventually, as mnemonic and problem-oriented programming languages were developed, the manufacturers distributed assemblers and compilers as part of their basic system support packages.

A distinguishing characteristic of these early programming support packages was the absence of any integration of individual support modules. Each module was a stand-alone entity which operated independently and did not interface with any of the other modules. And even though a few manufacturers provided minimal capabilities to perform serial job-to-job transition, the control languages were simply a series of delimiter statements that separated each job from the next and each program from its data.

The significant feature of second generation systems was the development of an integrated system concept that provided basic services to executing programs while maintaining control of job-to-job transition. This in turn introduced the concept of a main storage resident system that monitored system activity. Further, the support programs were no longer independent entities but became integrated modules that could be called and executed as required. Consequently, the term operating system was formally

introduced at this time to distinguish the software that provided support functions from an installation's application-oriented software.

Input and output device control was the most commonly implemented support feature of these systems. This service eliminated much of the programming previously required for I/O error conditions and provided additional facilities such as record blocking/deblocking and label checking. Other services included capabilities to request full and partial core dumps, to load subroutines dynamically, and to maintain inter-job communication regions. Further, the concept of on-line libraries for executable programs and subroutines was introduced and much greater flexibility was developed in the area of job-to-job transition. For example, the libraries gave rise to the multi-step job whereby programs residing in an on-line library could be interspersed and/or combined with programs in the input stream to accomplish a series of given tasks.

Finally, the controlling system language ceased to be a composite of delimiter and sentinel cards and became, in fact, a control language. The concept of an input job deck was developed whereby both job and user were identified, I/O device assignments enumerated, and program-execution sequence specified. Interspersed in the job deck could be the input data required for each job step, along with sentinels to prevent application programs from reading subsequent control cards. These operating systems nevertheless retained the serial processing orientation of first-generation systems.

Current third-generation operating systems, while retaining most of the features of earlier systems, have expanded total system control so that serial job-to-job transition represents only one of many possible operating system functions. The most significant advance made by third-generation systems has been the development of a capability to schedule and execute jobs in other than a sequential or serial manner. This capability, called multiprogramming, not only allows the concurrent execution of two or more programs, but permits execution scheduling to be based upon such factors as resource availability and job priority rather than strictly upon job input sequence.

In order to provide these, as well as other, capabilities, third-generation systems have assumed complete control of the computer system and inhibit executing application programs from performing any action that could degrade system performance or interfere with system monitoring. Unlike second-generation systems, which relied upon programming conventions to maintain system integrity, third-generation systems make use of hardware features which physically prevent application programs from executing certain types of instructions

or from utilizing the core areas assigned to other executing programs or to the resident supervisor. Thus, while earlier operating systems could be purposefully circumvented by the knowledgeable programmer (or perhaps inadvertently by the novice), third-generation system hardware makes this much more difficult, if not impossible.

However, the most distinguishing characteristics in the development of third-generation systems has been the emergence of the operating system as a necessity rather than an aid. The typical third-generation system not only controls the scheduling and execution of application programs but also performs all I/O operations, fields and processes all machine and program error conditions, and supervises or invokes recovery and restart capabilities when needed. Consequently, the scope of control has expanded well beyond the initial job management facilities originally associated with operating system technology and incorporates almost total control of both the hardware and software environment.

Section 3 presents a detailed and comprehensive description of contemporary operation system functions. The reader is therefore referred to that section for a further exposition of operating system components and features.

2.2 Types of Third-Generation Operating Systems

In discussing contemporary operating systems, it is convenient to have a breakdown of different systems types in order to isolate certain characteristics which may be common to members of one group and not to others. Ideally, the grouping should be relatively exclusive while permitting straightforward and easy categorization.

Since third-generation operating systems characteristically support a number of different system applications, the type of application supported would initially appear to be useful in discussing system characteristics. For example, the following major applications are supported by contemporary systems:

- batch processing,
- real-time processing,
- communications-oriented processing, and
- interactive terminal usage.

Unfortunately, there is enough diversity among contemporary commercial operating systems in the amount of support provided to each area to make such a categorization awkward. Many systems fail to make any provision within the supervisor for one or more

of these applications; others provide only minimal support for some and yet provide extensive support for other applications. Consequently, if this were the classification scheme, a system might be described as full batch processing, extensive real-time processing, partial communications support, with no interactive capability. While this description provides an excellent picture of the particular system, it also creates a potentially large number of different system categories. Furthermore, the terms -- full, extensive, partial and minimal -- are too likely to reflect the cataloger's biases.

On the other hand, most systems employ one of three basic design approaches: serial processing, multiprogramming, or time-sharing. Although these categories are somewhat too broad (and to a degree, overlapping), they do provide a basis for ascribing certain characteristics to various types of operating systems.

These design approaches may be illustrated by the manner in which a primary system resource, main storage, is allocated to application programs. When this resource is totally allocated to one application program at a time, the system is called a serial processing system. When main storage may be shared by two or more application programs at one time, the system is a multiprogramming system. A time-sharing system is identified by the fact that several programs may be allocated the same area of main storage for interleaved program execution.

With these storage allocation modes in mind, five systems types were selected to categorize the operating systems analyzed. Since the preponderance of the systems observed provide multiprogramming facilities, a further distinction was made concerning the degree of multiprogramming support available and the type of time-sharing service offered. This distinction permitted the isolation of several attributes which, while not common to every multiprogrammed system, created major subsets in the area of system design. The five systems types selected were:

- serial processing systems,
- basic multiprogramming systems,
- extended multiprogramming systems,
- serial time-sharing systems,
- multiprogramming time-sharing systems.

The simplest type of system is the serial processing system. This system is designed predominately for batch processing applications and provides simple job-to-job transition for consecutive programs in the input stream. This type of system does not permit any concurrency of operation either from a multiprogramming or time-sharing standpoint.

The multiprogramming systems cover a wide range of sophistication from rather small real-time-oriented processors to large batch and interactive multiprocessor systems. The definitive characteristic of the multiprogramming system is the concurrent core residence and interleaved execution of two or more programs. Many multiprogramming systems are designed strictly for single processors. However, some systems are designed to permit multiple processors, thereby achieving simultaneous as well as interleaved program execution.

A basic distinction can be made in the way storage is allocated to problem programs in a multiprogramming environment. If main storage is divided into fixed partitions or areas and each area is allocated only to jobs from a separate input stream or device, the system is a basic multiprogramming system (e.g., IBM's DOS/360, RCA's TDOS, and XDS's Batch Processing Monitor). In contrast, if separate input streams are merged on secondary storage prior to scheduling a job, and main storage is allocated to the first program capable of using the area, then the system is an extended multiprogramming system (e.g., IBM's OS/360, GE's GECOS III, and UNIVAC's EXEC-8).

This distinction can be further illustrated by the scheduling process. In a basic multiprogramming system, the scheduling process is external to the system. The area of main storage assigned to the program for execution is determined when the input device is selected to load the program. In an extended multiprogramming system, on the other hand, scheduling is internal. Although every program may not be able to execute in every partition (due to partition size, priority or other scheduling restrictions), the determination of the storage partition to be used is conceptually independent of the input device from which the program was entered. Thus, successive programs loaded from the same input device may execute concurrently in an extended system, but can only execute serially in a basic system. In many cases, a basic multiprogramming system may be thought of as a collection of several serial processing systems where each serial system operates in a uniquely assigned storage partition.

The time-sharing systems also permit several programs to execute concurrently. A varying time slice or time quantum is assigned to each program and the program is allocated the CPU and main storage for that period of time. At the end of the quantum, if another time-sharing program is waiting, the storage area and CPU may be reassigned to the next program. This usually involves removing the first program (and data) from main storage and storing it on an intermediate storage device. After all programs in the system have received separate execution time quanta, control is returned to the first program for another time quantum.

Serial time-sharing systems operate in much the same sequential manner as a serial processing system. All of the programs in execution are placed into a cyclic scheduling table and each program is activated in a fixed sequence. Some serial processing systems buffer the loading and unloading of the time-shared programs by using two or more main storage partitions. However, even though the system thus permits concurrent core residence of more than one program, if the actual execution sequence relies on cyclic rather than interleaved execution of these areas, then the system still has a serial orientation.

Multiprogramming time-sharing systems are those systems which provide the same time-sharing services mentioned above but, in addition, interleave these services with the execution of other programs. In some extended multiprogramming systems, one partition or storage area is devoted to time-sharing while the remaining areas process non-time-sharing applications. Consequently, systems with this orientation (GECOS III, EXEC-8) appear in both the multiprogramming time-sharing and the extended multiprogramming categories. Other systems are basic multiprogramming systems that have been extended to provide time-sharing services. Also, there are a few time-sharing systems that provide interleaved execution of core resident time-sharing programs in addition to the normal cyclic execution sequence. Many of these systems do not support the batch processing capabilities normally associated with a multiprogramming system (e.g., RCA/TSS, IBM/TSS). Thus, this category not only has a large degree of overlap with other categories, it also has a wide variety of different design approaches employed to provide time-sharing services in a multiprogramming environment.

Descriptions of the five system types are summarized in Table 2-1 and related to the operating systems of 12 major vendors in Table 2-2.

<u>OS TYPE</u>	<u>DEFINITION</u>
Serial Processing System	A system that provides core residence for a single application program. Program flow through the system occurs in a sequential manner.
Basic Multiprogramming System	A system that provides for concurrent core residence and processing of multiple programs. Execution occurs either simultaneously (multiprocessing) or by means of an interweaving process. Each partition or program area is independent of the others, is assigned its own unique input and output devices, and is scheduled by external control.
Extended Multiprogramming System	A multiprogramming system (as defined above) that has the additional capability of internal program scheduling. Input streams are queued on direct access devices so that the input readers serve all partitions. Programs are not necessarily confined to execution in any particular partition.
Serial Time-Sharing System	A system that provides for concurrent serial processing of multiple programs by allocating the system resources (including core) on a round-robin basis. A program resides on secondary storage, is brought into main storage to execute for a fixed time period (quantum) and then returned to secondary storage.
Multiprogrammed Time-Sharing System	A multiprogramming system where one or more of the partitions is assigned to process time-sharing programs.

Table 2-1 Definition of Operating System Types

BURROUGHS CORPORATION

B2500/B3500	C	Basic Control Program	Serial Processing
B2500/B3500	B	Master Control Program	Extended Multiprogramming
B5500	B	Master Control Program	Extended Multiprogramming
B5500	B	Master Control Program/ Time-Sharing System	Multiprogramming Time-Sharing
B6500	A	Master Control Program	Extended Multiprogramming

CONTROL DATA CORPORATION

31/32/33/3500	B	Mass Storage Operating System	Basic Multiprogramming
31/32/33/3500	B	Real-Time Scope	Basic Multiprogramming
3600	A	Scope Operating System	Serial Processing
3600	A	Drum Scope System	Basic Multiprocessing
3300/3500	A	Master Operating System	Extended Multiprogramming
6400/6500/6600	A	Scope 3	Extended Multiprogramming

DIGITAL EQUIPMENT CORPORATION

PDP-8	C	Disk Monitor System	Serial Processing
PDP-8	C	Time-Sharing System	Serial Time-Sharing
PDP-15	C	Advanced Monitor System	Serial Processing
PDP-10	B	Single User Manitor	Basic Multiprogramming
PDP-10	B	Background/Foreground Manitor	Basic Multiprogramming
PDP-10	B	Multiprogramming Monitor	Multiprogramming Time-Sharing
PDP-10	B	Swapping Monitor	Multiprogramming Time-Sharing

GENERAL ELECTRIC CORPORATION

600 Series	A	GECOS III	Extended Multiprogramming/ Multiprogramming Time-Sharing
------------	---	-----------	---

HEWLETT-PACKARD COMPANY

2000A	C	2000A System	Serial Time-Sharing
-------	---	--------------	---------------------

HONEYWELL, INC.

Series 200	C	MOD 1 Operating System	Basic Multiprogramming
Series 200	B	MOD 2 Operating System	Extended Multiprogramming
Series 200	A	MOD 4 Operating System	Extended Multiprogramming
Model 8200	A	MOD 8 Operating System	Extended Multiprogramming

C=Small Scale (less than 32K)

B=Medium Scale (32K - 132K)

A=Large Scale (greater than 132K)

Table 2-2. Operating

INTERNATIONAL BUSINESS MACHINES CORPORATION

1800	B	Multiprogramming Executive	Basic Multiprogramming
System 360	C	Basic Operating System	Basic Multiprogramming
System 360	B	Disk & Tape Operating System	Basic Multiprogramming
System 360	B	Operating System - PCP	Serial Processing
System 360	A	Operating System - MFT/MVT	Extended Multiprogramming
System 360	A	Operating System - MVT(TSO)	Multiprogramming Time-Sharing
System 360, 67	A	Time-Sharing System	Multiprogramming Time-Sharing

NATIONAL CASH REGISTER COMPANY

Century Series	C	B1 Operating System	Serial Processing
Century Series	C	B2 Operating System	Basic Multiprogramming
Century Series	B	B3 Operating System	Basic Multiprogramming

RCA CORPORATION

SPECTRA 70	C	Primary Operating System	Serial Processing
SPECTRA 70	C	Tape Operating System	Basic Multiprogramming
SPECTRA 70	B	Disk Operating System	Basic Multiprogramming
SPECTRA 70	B	Tape-Disk Operating System	Basic Multiprogramming
SPECTRA 70 (Model 46)	A	Time-Sharing System	Multiprogramming Time-Sharing

SYSTEMS ENGINEERING LABORATORIES, INC.

810A/810B	C	Operating System	Serial Processing
810B	B	Real-Time Executive	Basic Multiprogramming

UNIVAC DIVISION, SPERRY RAND CORPORATION

9300	C	Operating System	Basic Multiprogramming
9400	B	Operating System	Extended Multiprogramming
418-111	B	Real-Time Operating System	Extended Multiprogramming
494	A	Executive Omega System	Extended Multiprogramming
1108	A	EXEC-8	Extended Multiprogramming Multiprogramming Time-Sharing

XEROX DATA SYSTEMS, INC.

SIGMA 2	C	Basic Control Monitor	Basic Multiprogramming
SIGMA 2	C	Real-Time Batch Monitor	Basic Multiprogramming
SIGMA 5/7	B	Batch Processing Monitor	Basic Multiprogramming
SIGMA 5/7	A	Batch Time-Sharing Monitor	Multiprogramming Time-Sharing

Systems Analyzed in the OS Analysis Project

2.3 Operating System Levels

The degree of sophistication of an operating system generally corresponds to the amount of main storage required for its effective operation. System levels, therefore, have been established for all of the systems analyzed based on computer main storage size. The minimum storage requirements for each system have been largely ignored since these generally apply to a primitive version of the system which offers relatively few options and usually does not reflect the full capabilities of the system. Similarly, maximum sizes are not considered. While most operating systems will operate on the largest storage size available with the computer, they will not normally make effective use of the resources of the larger machine. Therefore, an attempt has been made to determine the size of main storage required to support the operating system and user programs in a normal operational environment. In general, three broad computer levels have been selected:

Level A: large scale computers with main storage in excess of 132K bytes,
(where a byte consists of 8 bits);

Level B: medium scale computers with main storage ranging from 32K
bytes to 132K bytes; and

Level C: small scale computers with main storage generally less than
32K bytes.

It should be noted that many computers are not byte oriented; however, due to the lack of common word size (in bits) among the computers analyzed, the 8-bit byte was arbitrarily chosen as the standard. The storage requirements for all operating systems were then stated in terms of the 8-bit byte. In this way, the main storage requirements of each system, regardless of the storage organization, could be compared roughly with others.

Table 2-2 presented previously also classified the operating systems of 12 vendors in terms of the system types and levels described above. It must be noted that the classification has been made on the basis of documentation supplied by the respective vendors. No attempt has been made to determine the effectiveness of each operating system in terms of performance or the degree to which each of the functions is actually implemented.

2.4 Operating System Functions

The analysis of contemporary operating systems was directed toward developing a functional classification structure to identify common operating system functions. Because of the wide variety of systems observed, a hierarchical classification scheme was selected as being the most convenient means of presenting both the general and the common capabilities along with specific implementation techniques used in various system designs. The following guidelines were developed to aid in determining the structure of the classification scheme:

- the structure must be oriented in such a way that it accurately describes the attributes of the wide variety of operating systems surveyed;
- the structure must account for varying sizes or scales of operating system support capability;
- the structure must be amenable to operating system validation in that the functions identified could become the items to be validated;
- the scheme must be flexible enough to accommodate new techniques and facilities as operating system technology advances;
- the scheme must be sufficiently broad to account for the variety of operational modes such as real-time, time-sharing, batch job processing, etc.

The resulting structure relied upon external or user-oriented functions at the upper levels of the hierarchy and internal or implementation-oriented functions at the more detailed lower levels of the hierarchy. To retain sufficient generality, certain functions that were actually system-dependent techniques, were not listed as subfunctions nor were they assigned a hierarchical level number within the structure. Instead, they were identified under the function to which they applied but were preceded by a bullet character (●) indicating that they were attributes, techniques or examples.

The following overview of the functional structure illustrates the categorization method employed:

Part I: Executive/Control Functions: The components of the operating system that maintain real-time execution control over the system environment.

- 1.0 Job Management: The real-time initiation, scheduling, monitoring and control of normal system operations together with the necessary allocation of resources.
- 2.0 Diagnostic Error Processing: The recognition of the occurrence of error conditions within the system along with the corresponding corrective actions.
- 3.0 Processing Support: The routines within the supervisor which accomplish a variety of miscellaneous real-time services for an application program.

Part II: System Management Functions: The non-real-time components of the operating system which support and maintain both system and application programs.

- 1.0 Operating System Management: The generation and maintenance of the computer operating system.
- 2.0 Program Maintenance: The support functions provided to the user for the maintenance and modification of application programs.
- 3.0 Compiler Interfaces: Operating system functions that provide supporting services to the system language compilers.
- 4.0 Management Support Utilities: Utility programs provided for the use of the system manager. These programs perform a wide variety of functions in support of initializing, testing, and monitoring the system.

Part III: Data Manipulation Functions: The components of the operating system that permit the user to access and process data. These functions may either be independent utility programs or subroutines incorporated within a user program.

- 1.0 Data Management: Comprehensive facilities which provide support for programmed access to the data files within the system. These facilities may take two forms: routines supporting application program access to the data base and an independent data management system supporting user access to the data-base.
- 2.0 Data Handling Utilities: Utility programs which provide a variety of independent services to manipulate and/or display various groupings of data elements.
- 3.0 Sorting and Merging: Those routines that sequence strings of records according to keys within each record.

In the structure, upper hierarchical levels are common to most operating systems. Lower hierarchical levels, as mentioned above, are not universally implemented. For example, one of the functions under job management is the function scheduling. All operating systems have a scheduling function, though in some it is implemented simply on a "first come, first served" basis. The next functional level under scheduling considers the types of scheduling functions available: algorithmic, time initiated, event initiated, and program initiated. Each is a separate form of scheduling, of which one or more may be available in any given system. However, since all forms are not equally available on all systems, that functional level marks the division between common functions and implementation-oriented functions.

It should also be recognized that while some system operations are germane to a unique functional category, many operations can be attributed to a number of different functional areas. The greater the level of detail within the hierarchy, the greater the possibility of overlap among functional categories. In an effort to minimize this overlap, while retaining a valid classification structure, decisions were made to allocate subfunctions to only one category where feasible even though the subfunction could legitimately be included elsewhere. Input/output error correction, for example, is categorized as a part of hardware error control rather than as a part of I/O control.

In some cases, overlap cannot be avoided. In these areas, an attempt has been made to identify the function at one point and identify it as an attribute or example elsewhere. The roll-out/roll-in feature whereby a program of low priority is removed from core to provide additional working space for a higher priority program is an example of this type of problem. Roll-out/roll-in control is defined as a subfunction of program loading. On the other hand, roll-out/roll-in is one of the possible attributes or methods of the function system initiated restarting.

The functional classification scheme is presented in the following pages in the established hierarchical format. Each level is a detailed functional breakdown of more general functions of a preceding higher level. Consequently, each lower level is identified by a more detailed hierarchical level number (e.g., 1.4.1.2 is a lower level within the hierarchy than 1.4.1).

Section 3 of this report presents a description of all of the major functions in this structure and illustrates how various types of systems implement them.

Operating System Functional Classification Scheme

Part I: Executive/Control Functions

- 1.0 Job Management
 - 1.1 Job Control
 - 1.1.1 Scheduling
 - 1.1.1.1 Algorithmic Scheduling
 - Priority Recognition
 - Resource Availability
 - 1.1.1.2 Time Initiated Scheduling
 - Elapsed Interval
 - Periodic Interval
 - Time of Day
 - 1.1.1.3 Event Initiated Scheduling
 - Interrupt Initiated
 - Unsolicited Inputs
 - 1.1.1.4 Program Initiated Scheduling
 - Subsequent Execution
 - Asynchronous Execution
 - 1.1.1.5 Conditional Scheduling
 - Prior Task/Job Completion
 - Prior Task Error Code
 - 1.1.1.6 Scheduling Queue Maintenance
 - 1.1.1.6.1 Single Input Queue Control
 - 1.1.1.6.2 Multiple Input Queue Control
 - 1.1.2 Resource Allocation
 - 1.1.2.1 Core Storage Allocation
 - 1.1.2.1.1 Fixed Block Allocation
 - 1.1.2.1.2 Dynamic Allocation
 - Buffer Pools
 - Free Core Pools
 - Program Expansion Area Pools
 - 1.1.2.1.3 Page Allocation
 - 1.1.2.1.4 Common (Shared) Core Allocation
 - 1.1.2.1.5 Core Storage Access Control
 - Storage Protection
 - 1.1.2.2 I/O Device Allocation
 - 1.1.2.2.1 Allocation by Specific Device
 - 1.1.2.2.2 Allocation by Device Type
 - Card Reader
 - Tape
 - Disk
 - 1.1.2.2.3 Allocation by Device Category
 - Sequential
 - Direct Access

Operating System Functional Classification Scheme (cont'd.)

Part I: Executive/Control Functions

- 1.1.2.2.4 Shared Device Allocation
 - Operator's Console/Display
 - Common I/O Files
- 1.1.2.2.5 Dynamic Allocation
 - Temporary Files
 - System Work Files
- 1.1.2.3 Common Subroutine Allocation
 - 1.1.2.3.1 Reentrant Routine Control
 - 1.1.2.3.2 Serially Reusable Routine Control
 - 1.1.2.3.2.1 Lock/Unlock Facilities
 - 1.1.2.3.2.2 Request Queue Control
- 1.1.3 Program Loading
 - 1.1.3.1 Structure Control
 - Paged
 - Overlay
 - Dynamic
 - 1.1.3.2 Loading Control
 - 1.1.3.2.1 Program Relocating
 - 1.1.3.2.2 Storage Compacting
 - 1.1.3.3 Swapping Control
 - 1.1.3.3.1 Roll Out/Roll In Control
 - 1.1.3.3.2 Paging Control
 - Static
 - Demand
- 1.1.4 Event Monitoring
 - 1.1.4.1 Dispatching Control
 - 1.1.4.1.1 Time Slicing Control
 - 1.1.4.1.2 Contention (Priority) Dispatching
 - 1.1.4.1.3 Dispatcher Queue Maintenance
 - 1.1.4.2 Event Synchronization
 - I/O Device Completion
 - Time Interval Interrupts
 - Sub-Task Execution/Completion
 - 1.1.4.3 Interrupt Processing Control
 - 1.1.4.3.1 Interrupt Priority Recognition
 - 1.1.4.3.2 Interrupt Masking/Disabling
 - 1.1.4.3.3 Interrupt Stacking
 - 1.1.4.4 Program Limit Monitoring
 - Output Record Limits
 - Execution Time Limits
- 1.1.5 Program Termination Processing
 - 1.1.5.1 Resource Deallocation
 - Closing Files
 - Releasing Devices
 - Releasing Core

Operating System Functional Classification Scheme (cont'd)

Part I: Executive/Control Functions

- 1.1.5.2 Summary Information Outputting
 - Record Counts
 - Run Times
 - Error Summarization
- 1.1.5.3 Abnormal Termination
 - Core Dumps
 - File Dumps
 - Error Codes
 - Program Recovery Initiation
- 1.2 I/O Control
 - 1.2.1 I/O Scheduling
 - 1.2.1.1 Device Resolution
 - 1.2.1.2 Request Stacking
 - 1.2.1.2.1 Device Queue Maintenance
 - 1.2.1.2.2 Channel Queue Maintenance
 - 1.2.1.2.3 I/O Initiation
 - Priority Recognition
 - Arm Optimization
 - 1.2.1.3 Alternate Routine Control
 - Alternate Channels to Device
 - Alternate Devices
 - 1.2.2 Data Transfer
 - 1.2.2.1 Buffering Control
 - 1.2.2.1.1 Buffer Pool Maintenance
 - 1.2.2.1.2 Buffer Handling
 - Simple Buffering
 - Exchange Buffering
 - Chained Segment Buffering
 - 1.2.2.2 Data Code Translation
 - Compressed Formats
 - Character Code Conversion
 - Paper Tape Formats
 - Teleprocessing Code Conversion
 - 1.2.3 Device Manipulation
 - Tape/Disk Positioning
 - Card Stacking
 - Page Ejecting
 - 1.2.4 Remote Terminal Support
 - 1.2.4.1 Interactive Communication Control
 - 1.2.4.2 Terminal-to-Terminal Communication Control
 - 1.2.4.3 Control of Remote Batch Job Initiation
- 1.3 System Communication
 - 1.3.1 System Startup
 - 1.3.1.1 System Initialization
 - 1.3.1.1.1 Batch Job Initialization
 - 1.3.1.1.2 Foreground Initialization
 - 1.3.1.1.3 Standard Option Modification

Operating System Functional Classification Scheme (cont'd.)

Part I: Executive/Control Functions

- 1.3.1.1.4 Resource Specification
 - Peripheral Devices
 - Partition Sizes
 - Input/Output Symbionts
 - 1.3.1.1.5 Parameter Specification
 - Date
 - Time
 - 1.3.1.2 System Restart
 - 1.3.1.2.1 Suspended Queue Resumption
 - 1.3.1.2.2 Parameter Respecification
 - 1.3.2 Job Control Communication
 - 1.3.2.1 Non-Interactive Control
 - Input Stream Control Cards
 - Cataloged Procedures
 - Operator Console Commands
 - 1.3.2.2 Interactive Control
 - On-Line/Remote Terminal Dialogue
 - Operator Console Dialogue
 - 1.3.3 Input/Output Stream Control
 - 1.3.3.1 Symbiont Processing
 - 1.3.3.2 Input/Output Queue Maintenance
 - 1.3.3.3 Control Command Analysis
 - 1.3.4 Resource Status Modification
 - 1.3.4.1 Addition/Deletion of System Resources
 - 1.3.4.2 Partition Size Modification
 - 1.3.4.3 Substitute Device Identification
 - 1.3.5 System Status Interrogation
 - 1.3.5.1 Current User Display
 - 1.3.5.2 Resource Status Display
 - 1.3.5.3 Job Status Display
- 1.4 Recovery Processing
 - 1.4.1 Checkpointing
 - 1.4.1.1 Program Initiated Checkpointing
 - 1.4.1.2 System Initiated Checkpointing
 - Roll Out/Roll In
 - 1.4.1.3 Externally Initiated Checkpointing
 - Input Stream Control Card
 - Operator Console
 - On-Line/Remote Terminal
 - 1.4.1.4 Checkpoint Notification
 - Operator Console
 - Job Output Log
 - 1.4.2 Restarting
 - 1.4.2.1 Program Initiated Restarting
 - 1.4.2.2 System Initiated Restarting
 - Roll Out/Roll In
 - Error Detection Occurrence

Operating System Functional Classification Scheme (cont'd.)

Part I: Executive/Control Functions

- 1.4.2.3 Externally Initiated Restarting
 - Input Stream Control Card
 - Operator Console
 - On-Line/Remote Terminal
- 1.4.2.4 Device Repositioning
 - Input Stream
 - Output Stream
 - Peripheral Devices
 - Teleprocessing Devices
- 2.0 Diagnostic Error Processing
 - 2.1 Hardware Error Control
 - 2.1.1 Error Correction
 - 2.1.1.1 Fault Analysis
 - 2.1.1.2 Event Retry/Retransmission
 - Retransmission Threshold
 - 2.1.1.3 Controlled Linkage to User Error Routines
 - 2.1.2 Error Notification
 - 2.1.2.1 Operator Notification
 - 2.1.2.2 Program Notification
 - 2.1.2.3 Device Error Statistic Accumulation
 - 2.1.2.4 Diagnostic Logout of Permanent Errors
 - 2.1.3 Error Recovery
 - 2.1.3.1 System Reconfiguration
 - 2.1.3.1.1 Alternate Device Utilization
 - 2.1.3.1.2 Controlled System Degradation
 - 2.1.3.2 Manual Reconfiguration
 - 2.1.3.3 On-Line Diagnostic Testing Control
 - 2.2 Program Error Control
 - 2.2.1 Error Correction
 - 2.2.1.1 Controlled Linkage to User Error Routines
 - 2.2.1.2 Simulation of Non-Hardware Implemented Operations
 - Non-Implemented OP Codes
 - 2.2.2 Error Notification
 - 2.2.2.1 Operator Notification
 - 2.2.2.2 Program Notification
 - 2.2.2.3 Diagnostic Error Log-Out
 - 2.2.3 Program Termination
 - 2.3 Interface Error Control
 - 2.3.1 Operator Key-In Editing
 - 2.3.2 Control Command Editing
 - 2.3.3 Remote Terminal Communication Editing
 - 2.3.4 Program-to-System Link Verification

Operating System Functional Classification Scheme (cont'd.)

Part I: Executive/Control Functions

- 3.0 Processing Support
 - 3.1 Timing Service
 - 3.1.1 Real-Time Clock Service
 - Date
 - Time of Day
 - 3.1.2 Interval Timer Service
 - 3.1.2.1 Scheduling Periodic Interrupts
 - Loop Control
 - Timing Analysis
 - 3.1.2.2 Temporary Task Suspension Control
 - 3.2 Testing/Debugging Service
 - 3.2.1 Storage Dump Control
 - 3.2.1.1 Snapshot Control
 - 3.2.1.2 Partial Dump Control
 - 3.2.2 Tracing Control
 - Data Tracing
 - Instruction Tracing
 - Logic Tracing
 - Supervisor Entry Tracing
 - 3.2.3 System Test Mode Control
 - 3.2.3.1 I/O Simulation
 - Error Simulation
 - I/O Re-routing
 - 3.2.3.2 Abnormal Termination Service
 - Storage Dumps
 - File Dumps
 - Subsequent Task Execution
 - 3.2.3.3 Interactive Testing Service
 - Breakpoints
 - Memory Searching
 - Memory Modification
 - Interrupt or Error Notification
 - 3.3 Logging and Accounting
 - 3.3.1 Maintaining Job Charge Information
 - 3.3.1.1 CPU Time Recording
 - 3.3.1.2 I/O Channel and Device Time Recording
 - 3.3.1.3 Resource Utilization Recording
 - 3.3.1.4 Controlled Linkage to User-Supplied Accounting Routines
 - 3.3.2 Maintaining Error Statistics
 - 3.3.2.1 Hardware Error Summary Accumulation
 - 3.3.2.2 Program Error Summary Accumulation
 - 3.3.2.3 Hardware Log-out Storage and Maintenance
 - 3.3.2.4 Error Statistic Retrieval

Operating System Functional Classification Scheme

Part I: Executive/Control Functions

- 3.3.3 Maintaining System Utilization Statistics
 - 3.3.3.1 User Account Summary Recording
 - 3.3.3.2 File Access Summary Recording
 - 3.3.3.3 System Service Request Recording
 - 3.3.3.4 System Performance Monitoring
- 3.4 Program Accessible System Description Maintenance
 - 3.4.1 Current System Status Interrogation
 - Number of Other Users
 - Core Storage in Use
 - Device Status
 - Elapsed Program Execution Time
 - 3.4.2 System Definition Interrogation
 - System Components
 - Maximum Number of Users
 - Generation Options Selected

Operating System Functional Classification Scheme

Part II: System Management Functions

- 1.0 Operating System Management
 - 1.1 System Generation
 - 1.1.1 Initialization
 - 1.1.1.1 Program Media Transcription
 - Bootstrap Loaders
 - Magnetic Tape-to-Disk Copy Routines
 - 1.1.1.2 Workspace Allocation
 - Volume Preparation
 - 1.1.1.3 Generating Configuration Identification
 - 1.1.2 Supervisor Generation
 - 1.1.2.1 Operating Configuration Identification
 - Memory Size
 - Partitions
 - Number of CPU's and I/O Controllers
 - I/O Devices
 - Console Devices
 - 1.1.2.2 Installation Parameter Specification
 - Memory Protection
 - Privacy Codes
 - Error Recovery Options
 - Accounting Options
 - 1.1.2.3 Foreground Processing Specification
 - Interrupt Assignments
 - Task Scheduling Priorities
 - Dynamic Core Allocation Pool
 - 1.1.2.4 Background Processing Specification
 - Batch Processing Specifications
 - Time Sharing Options
 - 1.1.2.5 Resident/Transitional Module Determination
 - 1.1.3 Operating System File Creation
 - 1.1.3.1 Support System Program Selection/Tailoring
 - Compilers
 - Data Management Systems
 - Utilities
 - 1.1.3.2 User-Developed Program Inclusion
 - 1.1.4 Authorized User Declaration
 - User Identification
 - Passwords
 - Accounting Controls
 - Resource Limitations
 - Priorities

Operating System Functional Classification Scheme (cont'd.)

Part II: System Management Functions

- 1.1.5 Default Specification
 - Standard Options
 - Priorities
- 1.2 System Maintenance
 - 1.2.1 Dynamic Maintenance
 - 1.2.1.1 System Reconfiguration Control
 - 1.2.1.1.1 Recognition of Added Processing Elements
 - 1.2.1.1.2 Recognition of Deleted Processing Elements
 - 1.2.1.2 Fallback (Degraded Operation) Mode Control
 - 1.2.2 System Software Modification
 - 1.2.2.1 System Regeneration
 - 1.2.2.2 Module Addition/Replacement
 - Permanent Usage
 - Temporary Usage
 - 1.2.2.3 Patching
 - On-line Patching
 - Permanent Patching
 - Temporary Patching
 - 1.2.3 User Maintenance
 - 1.2.3.1 Defining New Commands
 - 1.2.3.2 Renaming Commands, Operands, Expressions, Values
 - 1.2.3.3 Alteration of User Default Values
- 2.0 Program Maintenance
 - 2.1 Library and Directory Maintenance
 - 2.1.1 Dynamic Cataloging
 - Load Modules
 - Task/Procedure Definitions
 - 2.1.2 Static Cataloging
 - Load Modules
 - Relocatable Modules
 - Source Modules
 - Macro Routines
 - Task and Job Procedures
 - 2.1.3 Utility Functions
 - 2.1.3.1 Copying
 - 2.1.3.2 Renaming
 - 2.1.3.3 Allocating/Deallocating/Repacking Library Space
 - 2.1.3.4 Punching/Listing/Displaying
 - 2.2 Load Module Generation
 - 2.2.1 Program Binding
 - 2.2.1.1 Program Element Relocating
 - 2.2.1.1.1 Dynamic Binding
 - Relocatable Loaders
 - 2.2.1.1.2 Static Binding
 - Linkage Editors
 - Collectors

Operating System Functional Classification Scheme (cont'd.)

Part II: System Management Functions

- 2.2.1.2 Input Module Control
 - Relocatable Modules
 - Executable Modules
- 2.2.1.3 Code Altering Facilities
 - Patching
 - Debug Support
- 2.2.2 Linkage Resolution
 - 2.2.2.1 Program Control Reference Resolution
 - 2.2.2.2 Data Field Reference Resolution
 - 2.2.2.3 Library Scan for Unresolved References
- 2.2.3 Multiple Structure Support
 - Overlay Programs
 - Segmented Programs
- 3.0 Compiler Interfaces
 - 3.1 Executive Routine Support
 - 3.1.1 Recognition of Compiler Parameters on OS Control Cards
 - Listing Options
 - Conditional Compilation Parameters
 - Input Source
 - Output Form
 - 3.1.2 System Maintained Compiler Communication Tables
 - 3.1.3 Input and Output File Control
 - Compile File Maintenance
 - 3.1.4 Program Testing/Debugging Control
 - 3.2 Library Support
 - Source Program Libraries
 - Macro Libraries
 - Subroutine Libraries
 - 3.3 System Utility Program Support
 - 3.3.1 Sort/Merge Linkage Support
 - 3.3.2 Peripheral Conversion Linkage Support
 - 3.3.3 Data Management System Linkage Support
- 4.0 Management Support Utilities
 - 4.1 Peripheral Device Support
 - 4.1.1 Volume Preparation
 - 4.1.1.1 Record/Track Formatting
 - 4.1.1.2 Directory Creation
 - 4.1.1.3 Space Allocation
 - Dynamic Communication Buffer Areas
 - 4.1.2 Volume Maintenance
 - 4.1.2.1 Diagnostic Verification
 - 4.1.2.1.1 Surface Analysis
 - 4.1.2.1.2 Track Replacement

Operating System Functional Classification Scheme (cont'd.)

Part II: System Management Functions

- 4.1.2.2 File Purging
 - Clear Core
 - Clear Disk/Drum
 - Erase Tape
- 4.2 System Simulation Routines
 - 4.2.1 System Facilities
 - I/O Device Activity
 - Real-Time Interrupts
 - 4.2.2 Communication Facilities
 - Message Transmission
 - Exception Processing
- 4.3 System Measurement Routines
 - Sampling Routines
 - Intercept Routines
- 4.4 Stand-Alone Utilities
 - 4.4.1 Status Display
 - Core and File Dumps
 - Diagnostic Logout Displays
 - 4.4.2 Recovery Support
 - Rebuild System Queues
 - Rebuild Message Queues
 - Reinitiate Suspended Processing

Operating System Functional Classification Scheme

Part III: Data Manipulation Functions

- 1.0 Data Management
 - 1.1 File Management Facilities
 - 1.1.1 File Location Recognition
 - 1.1.1.1 File Cataloging Control
 - Addition/Deletion of Cataloged Files
 - Catalog Determination of File Location
 - 1.1.1.2 Label Recognition
 - 1.1.2 File Access Control
 - 1.1.2.1 File Security Control
 - User ID Protection
 - Password Protection
 - Recognition of Hardware Keys
 - 1.1.2.2 Read/Write Access Control
 - Read Access Only
 - Selective Write Access
 - 1.1.2.3 Concurrent Access Control
 - Multi-User Read/Single-User Write Access
 - Record Lockout
 - 1.1.3 Backup and Restoration Facilities
 - 1.2 I/O Support Facilities
 - 1.2.1 Data Access Control
 - 1.2.1.1 Sequential Access Control
 - 1.2.1.2 Keyed/Indexed Access Control
 - 1.2.1.3 Random Access Control
 - 1.2.1.4 Teleprocessing Access Control
 - 1.2.1.4.1 Priority Message Control
 - 1.2.1.4.2 Message Queue Maintenance
 - 1.2.1.4.3 Input Message Decoding and Routing
 - 1.2.1.4.4 Output Message Routing
 - 1.2.1.4.5 Polling Control
 - 1.2.2 Data Blocking/Deblocking Control
 - 1.2.2.1 Record Acquisition
 - Locate Mode
 - Move Mode
 - 1.2.2.2 Record Type Support
 - Fixed Length
 - Variable Length
 - Undefined Length
 - 1.2.3 Label Processing
 - 1.2.3.1 Label Generation
 - System Standard Labels
 - User Standard Labels
 - User Non-Standard Labels

Operating System Functional Classification Scheme (cont'd.)

Part III: Data Manipulation Functions

- 1.2.3.2 Label Checking
- 1.3 Data Management System Facilities
 - 1.3.1 Control Specification
 - 1.3.1.1 Specification Interpretation
 - File Descriptive Tables
 - Query Descriptions
 - Report Descriptions
 - 1.3.1.2 Generation of Supporting Functions
 - Interpretative Tables
 - Executable Subroutines
 - 1.3.2 Data File Generation and Maintenance
 - 1.3.2.1 Structure Definition
 - Sequential
 - Hierarchial
 - Indexed
 - Ring
 - List
 - 1.3.2.2 Space Allocation
 - 1.3.2.3 Input Transaction Processing
 - 1.3.2.3.1 Input Format Definition
 - Pre-Defined Data
 - Self-Defining Data
 - 1.3.2.3.2 Input Validation
 - Range Verification
 - Specific Character Examination
 - Sequence Checking
 - Comparison Testing
 - 1.3.2.3.3 Input Alteration
 - Modification of Data Element Size
 - Encoding/Decoding of Data Elements
 - 1.3.2.3.4 Input Termination Recognition
 - Embedded Control Fields
 - Special Character (s)
 - 1.3.2.4 Logical File Maintenance
 - Single Value Updating
 - Record Selectivity
 - Subordinate File Updating
 - 1.3.2.5 Interactive File Maintenance
 - 1.3.2.6 File Reorganization
 - Restructuring
 - Merging
 - 1.3.2.7 Data Error Procedures
 - Interactive
 - Pre-Defined

Operating System Functional Classification Scheme (cont'd.)

Part III: Data Manipulation Functions

- 1.3.3 Data Qualification and Retrieval
 - 1.3.3.1 Retrieval Mode Control
 - 1.3.3.1.1 Interactive Querying
 - Programmed Queries
 - Cue-Response Queries
 - 1.3.3.1.2 Batched-Mode Queries
 - Prestored Queries/Skeletal Queries
 - 1.3.3.2 Query Processing
 - 1.3.3.2.1 Relational Operator Control
 - Boolean
 - Quantitative (Occurrence)
 - Arithmetic or Statistical
 - Application-Defined
 - 1.3.3.2.2 Term Resolution
 - Constant Value
 - Second Data Field
 - Arithmetic Expression
 - Interim Result
 - 1.3.3.3 Data Record Selection
 - Single File Retrieval
 - Multi-File Retrieval
 - Inter-File Retrieval
- 1.3.4 Data Output
 - 1.3.4.1 Output Media Control
 - Printed Listings
 - Punched Cards
 - Tape Files
 - CRT Displays
 - 1.3.4.2 Output Mode Control
 - 1.3.4.2.1 User-Structured Report Control
 - 1.3.4.2.2 System Defined Report Control
 - 1.3.4.2.3 Interactively Defined Report Control
 - 1.3.4.3 Output Format Control
 - 1.3.4.3.1 Labeling
 - Data Labels
 - Page Headings
 - Trailer Information
 - 1.3.4.3.2 Data Formatting
 - Positioning
 - Altering and Decoding
 - Counting/Totaling
 - 1.3.4.3.3 Pagination Control
 - Printed/Typed Reports
 - CRT Displays

Operating System Functional Classification Scheme (cont'd.)

Part III: Data Manipulation Functions

- 2.0 Data Handling Utilities
 - 2.1 Display Facilities
 - 2.1.1 Unformatted Display
 - Card to Printer
 - Tape to Printer
 - 2.1.2 Formatted Display
 - File Edits
 - Formatted Tape/Disk Dumps
 - Core Dumps
 - Directory Lists
 - 2.2 Peripheral Device Support
 - 2.2.1 Volume Positioning
 - 2.2.1.1 Forward/Backward Spacing
 - 2.2.1.2 Record/File Selection
 - 2.2.2 Media Copy Facilities
 - 2.2.2.1 Format Conversion
 - 2.2.2.2 Code Conversion
 - BCD to Baudot Code
 - Hollerith to Binary
 - 2.2.2.3 Field Insertion
 - Constant Values
 - Sequence Number
 - 2.2.2.4 Multi-Device Support
 - Card
 - Magnetic/Paper Tape
 - Random Access
 - Remote Terminal
 - Main Storage
 - 2.2.2.5 Dump and Reload Facilities
 - File/Storage Compaction
 - Overflow Area Elimination
 - Backup File Creation
 - 2.2.3 Data Editing Facilities
 - File Compare
 - File Scanning
 - 2.2.4 Test Data File Support
 - Test File Generation
 - Control Message Generation
- 3.0 Sorting and Merging
 - 3.1 Sort Module Development
 - 3.1.1 Control Card Editing/Interpreting
 - 3.1.2 Storage Requirement Determination
 - Core
 - Intermediate Storage
 - 3.1.3 Program Generation
 - Tailoring General Purpose Packages
 - Generating Unique Sort Modules

Operating System Functional Classification Scheme (cont'd.)

Part III: Data Manipulation Functions

- 3.1.4 Optional Support
 - Ascending/Descending Output Sequences
 - Single/Multiple Sort Control Fields
 - Field Type Recognition (Alphanumeric, Binary, Zoned/Packed Decimal, Floating Point, etc.)
 - User Specified Collating Sequences
 - Multi-Media Storage Devices (Magnetic Tape, Mass Storage, Random Access Devices)
- 3.2 Sort Module Execution
 - 3.2.1 Input Data Control
 - User-Generated Records
 - Established Data Files
 - Internal Record Address Table
 - 3.2.2 Sort Block Creation
 - Full Data Records (Record Sort)
 - Sort Key and Record Address (Tag Sort)
 - Sort Key and Selected Fields (Field Select Sort)
 - 3.2.3 Sequenced String Creation
 - Replacement/Selection Sort
 - Tournament Sort
 - Balanced Merge
 - 3.2.4 Overflow Control
 - Develop Sort Subfields
 - Spillover to Spare Device
 - 3.2.5 Sequenced String Merge
 - Polyphase Sort
 - Oscillating Sort
 - Cascade Sort
 - 3.2.6 Final Pass Control
 - 3.2.6.1 Multi-File Merging
 - 3.2.6.2 Final Pass Sequence Check
 - 3.2.6.3 Output Data Control
 - 3.2.6.3.1 Record Deblocking
 - 3.2.6.3.2 Record Address Table Creation
 - 3.2.7 User Routine Linkage Control
 - Label Processing
 - Input Record Insertion/Modification/Deletion
 - Output Record Insertion/Modification/Deletion
 - I/O Error Processing

SECTION III

DISCUSSION OF OPERATING SYSTEM FUNCTIONS

3.1 Introduction

Section II presented a hierarchy of computer operating system functions. This Section contains a tutorial discussion of the major functions in that hierarchy. The discussion covers the purpose of each function and a cross section of methods for implementation. It is assumed that the reader has some knowledge of the application and use of computer operating systems. However, the presentation is structured such that a prior technical knowledge of operating system design is not required and a glossary is presented in Appendix I for those terms not defined in context.

3.2 Discussion

The presentation is structured in the same format as the functional classification scheme and each operating system function is identified by the appropriate hierarchical level number. Within each part, the discussion is presented at a level where the functions are common to most operating systems. Thus, the executive/control job management functions are presented at the third hierarchical level, while the remaining executive control functions and all of the system management and data manipulation functions are presented at the second hierarchical level.

The major functional areas are presented on the following pages:

Part I	Executive/Control Functions	p. 35
Part II	System Management Functions	p. 55
Part III	Data Manipulation Functions	p. 62

Part I: Executive/Control Functions

1.0 JOB MANAGEMENT

The job management function is responsible for the initiation, scheduling, monitoring, and control of system operations for all jobs submitted to the system. In this context, a job encompasses all of the programs required for the execution of a given application. The job management subfunctions consist of: job control, input/output control, system communication, and recovery processing.

1.1 Job Control

In processing a job, the supervisor is actually concerned with independent job elements called tasks. The task (or, as it is sometimes called, the process or activity) is the basic unit of work within the operating environment. The operating system job control functions are those functions that regulate the use of the system resources by the various tasks comprising each separate job.

1.1.1 Scheduling

The purpose of the scheduling function is to select a job which is available for processing and prepare it for execution. The function may be extremely complex, as in an extended multiprogramming system where several jobs may be simultaneously available for execution, or quite simple, as in serial processing systems where the order of programs in the input stream may dictate the execution sequence.

Each mode of system operation (batch, real-time, or time-sharing) requires a distinct scheduling philosophy. The most straightforward method, assigning priorities to each type of event that may produce an external signal to the computer, is generally found in a real-time environment. When any event occurs, if a higher priority event is currently being processed, this event remains pending until that processing terminates. On the other hand, if a lower priority event is being processed, this processing is suspended and the system resources are made available to the interrupting event. The programs which process the real-time events may be permanently resident in core or may be loaded dynamically. In several systems designed primarily for real-time processing (e.g., the XDS Sigma 2/5/7, IBM 1800, etc.), this scheduling function is performed by the system hardware. Other systems require a software scheduling routine to provide this function in lieu of the hardware capability.

Scheduling for time-sharing systems is also relatively straightforward. When a time-sharing user logs-in on his console, the scheduling routine determines whether the system is currently saturated. If it is saturated, the user is not allowed to proceed; if not, he is immediately placed in an execution mode. Some systems require that the user identify his resource requirements at log-on time and will not schedule the user until all these resources are available. Other systems do not include this feature within the scheduling routine. Instead, if the user attempts to use a committed resource, he is suspended until that resource becomes free.

The greatest variation in the implementation of the scheduling facility exists in the handling of batch processing. The most elementary approach is to schedule each job for execution in the sequence of its occurrence in the input stream. When a system has separate input streams for several processing areas, as in serial processing or

basic multiprogramming systems, each input stream serves as a scheduling queue which is external to the system.

Further sophistication of the sequential approach is achieved by pre-reading the entire input stream and storing it on a secondary storage device such as a disk. By so doing, jobs may be executed in an order other than input stream sequence. In this type of system, scheduling parameters are introduced to control the execution sequence. Normally, these parameters are priorities, where a higher priority job is executed before a lower priority job. Alternatively, the parameters may be clock times, where a job is initiated at a selected time or is initiated to be completed by a certain time. Clock-time scheduling may also be used to initiate selected real-time jobs.

The batch scheduling philosophy of an extended multiprogramming system allows jobs submitted from more than one input stream to compete for execution assignment. Under this philosophy a scheduling queue on an intermediate storage device is mandatory, and jobs are placed on this queue whenever they are entered from either a local or remote input terminal. In this type of system, an input stream symbiont is used to read the input stream and store it on the scheduling queue. The symbiont is itself scheduled by an external event such as an operator or user command to initiate input stream processing.

Other scheduling capabilities available in various systems permit algorithmic, program-initiated, and conditional scheduling. Conditional scheduling permits the user to specify scheduling criteria which must be satisfied before the program can be scheduled. Criteria may be the presence or absence of certain errors in a previous job step or the setting of internal switches by another task. Program-initiated scheduling permits an executing program task to request that another program task be scheduled for either asynchronous or subsequent execution. Algorithmic scheduling is an extension of the priority scheduling concept where many factors will influence the selection of the next program chosen for execution. The relationship of these factors to one another is known as the scheduling algorithm. Some commonly used factors are: estimated run time, resource availability, job priority, CPU-I/O relationship, and the length of time an element has been in the queue. A few of the more sophisticated systems permit the operator to modify the scheduling algorithm during system operation.

Since most systems handle more than one type of processing mode, different scheduling philosophies are used for the real-time, time-sharing, and batch processing applications. Frequently, the lack of a processing workload in one of these modes will permit the scheduling mechanism of another mode to utilize normally reserved facilities. For example, the lack of current real-time or time-sharing processing may be recognized by the batch processing scheduler which would then initiate more batch processing jobs than under normal conditions. Similarly, a heavy real-time workload may force the suspension of some or all batch processing jobs so that core may be assigned to the more critical routines.

1.1.2 Resource Allocation

The resource allocation function is responsible for assigning resources to each executing job in such a way that conflicting resource assignments are avoided.

In general, system resources may be discussed in terms of CPU time, core storage, I/O devices, and information files. The allocation of CPU time to a program is called dispatching and is covered separately under Part I, Section 1.1.4. The other three areas are discussed below.

In a serial processing system, resource allocation routines do not exist to any significant extent since all of the system resources are normally made available to the executing program. In a multiprogramming or time-sharing environment, however, two or more programs may be executing concurrently and can produce conflicts if common resources are not controlled. Because of this problem, many scheduling algorithms require that all needed resources be assigned prior to scheduling the program. When this requirement is levied, each job must provide information to the scheduler describing the resources it will need during execution. These are usually stated as job control language parameters in the input stream or are contained in generated data blocks produced by the program binding process.

Some systems permit dynamic resource allocation. This allocation method allows an executing program to request the use of a resource only when it is needed. If the resource is uncommitted, it will be assigned to the program; otherwise, the program will be suspended until the resource becomes available. Dynamic resource assignment is usually found only in extended multiprogramming systems and is primarily advantageous where a large number of programs are executing concurrently. Since resources are only committed when they are being used rather than for the duration of an entire job, the technique may significantly reduce the total number of resources required for operating efficiency when compared to an environment using static allocation.

Core storage - There are four basic methods for allocating core storage in a multiprogramming environment. The first method involves the static assignment of fixed foreground and background processing areas. The foreground processing areas are generally allocated to real-time or time-sharing processing, while the background area is generally allocated to batch processing. Characterizing this form of storage organization is the fact that program priority is determined by the storage assignment: foreground areas always have execution priority over background areas.

The second method of storage allocation involves a series of fixed independent partitions of varying sizes. Each partition may have a separate input stream, or programs may be assigned to the smallest available partition that will contain them. In the latter case, the system may also provide parameters that prevent some programs from being assigned to certain partitions. A distinction between real-time, time-sharing, or batch processing is not provided in terms of partition definition.

The third method involves maintaining all free core as a large pool of storage. Each program is assigned the exact amount of storage that it requires from the pool and, upon terminating, returns the used storage to the pool. If time-sharing operations are undertaken in this environment, a block of core is normally removed from the pool to handle all of the time-shared applications.

In the fourth method, core is segmented into a series of fixed pages of relatively small size and these pages are maintained in a pool. Likewise, program instructions and data are segmented into pages. A program may then be assigned the number of pages it requires. The assigned pages need not be contiguously located in main storage. Many advanced systems keep unused pages of programs on a fast access drum and, during program execution, they are shuffled in and out of core as needed by a process called paging (described in Part I, Section 1.1.3).

Once core has been allocated to a program, additional core may be required for buffers, program expansion, etc. For these requirements, most systems utilize special purpose storage pools which are independent of normal allocatable storage. When an executing program requires storage in excess of its original allocation, it can request an additional allocation from the supervisor. The pools are scanned for an area of the proper size and when found, the area is assigned to the program. The area is returned to the pool either explicitly by the executing program when it has finished using it or implicitly when the program terminates.

Normally, storage protection mechanisms are also provided to prevent a program from modifying or accessing core storage areas outside its allocated regions. Conversely, many systems also provide common areas of core storage which may be shared by several executing programs.

I/O devices - In the background/foreground mode of storage assignment, I/O devices tend to be permanently allocated to one of the background or foreground partitions such that the allocation may be altered only by the console operator. Some systems in this class, predominantly real-time process control systems, do not control I/O device assignments at all and rely on programming conventions to avoid conflicting utilization.

In partitioned and paged processing environments, I/O devices may be allocated statically at job initiation time or dynamically during job execution. In general, systems oriented to the use of unit record or serial devices rely upon static assignment. On the other hand, systems oriented to the use of random access and teleprocessing devices are most likely to use dynamic assignment techniques since several programs may access a common device concurrently.

Information files - The information files available to a system consist of files of data and libraries of programs or subroutines. The allocation of each of these facilities is handled somewhat differently.

Data files may be statically assigned to a program during the scheduling process and treated in much the same way as an I/O device assignment. In such systems, the data files may not be assigned to a second program until the first program has terminated. Other operating systems are designed to permit access to the same data file by a number of concurrently executing programs. In this type of system, the file is dynamically allocated based upon the type of access the program requests. Read-only access requests are normally granted unless the data file is being modified. Write access requests are normally held in a pending state until all current read access requests have been satisfied. In a great many systems, only one program is allowed write access whereas any number of programs may be

allocated concurrent read access. Operating systems of this type also usually feature some form of security control that must be satisfied before file access is permitted. This aspect is described in greater detail in Part III, Section 1.1.2.

Programs or subroutines that can be used by more than one program are frequently designed to be loaded and executed when needed rather than incorporated as a part of the executing program during the binding process. These routines may be loaded into a specially-designated transient execution area, into any available area of core the executing program can find, or, if used frequently enough, made a part of the resident system. Allocation of these routines to requests is almost always dynamic rather than static.

There are three types of subroutine organization that require different allocation procedures. Non-reusable subroutines are subroutines that must be loaded "fresh" each time a request is made for the routine. Serially re-usable routines need not be reloaded; however, the routine, because of possible modification of constants or instructions, can only be used by one program at a time. Re-entrant routines may be used by any number of executing programs simultaneously. Thus, allocation of re-entrant routines is straightforward, whereas serially reusable routines must have a lock/unlock facility which limits their assignment to a single program at a time.

1.1.3 Program Loading

Loading Control - Once a job has been scheduled for execution and the requested resources have been assigned, the operating system loads the initial task of the job into the core area assigned for job execution. The system obtains the required load modules (see Part II, Section 2.2) either from a system library, a user library, or a temporary load module file.

Some systems only provide facilities for absolute loading. In this design, a program is loaded into a single, fixed main storage location which may have been determined when the load module was generated. However, most systems permit relocatable loading which allows a program to be loaded into any given storage location. Address resolution is usually accomplished by setting a base address register to the initial address of the loaded area. In a few systems, the program may be further relocated during execution by moving it to another area and changing the contents of the base address register.

Structure control - Operating systems may support several different kinds of program loading structures. The more common structures are categorized as simple, overlay, paged, and dynamic.

A simple program structure consists of a single module occupying a fixed amount of main storage. An overlay program allows for repeated use of the same blocks of core storage during different stages of program execution. Thus, the overlay program is segmented and each segment is loaded as it is required.

Paged and dynamic structures are considerably more sophisticated in scope and, consequently, are common only in the larger multiprogrammed and time-sharing environments. Dynamic loading allows a relocatable module to be loaded into main storage

upon a request for that module by an executing program. It differs from an overlay structure in that the area of main storage need not be prespecified, nor does the module necessarily replace or overlay an existing program segment. One of the major problems with dynamically-loaded programs is that the process creates unused core fragments. The unused core fragments result when all dynamically-loaded programs do not terminate concurrently. Consequently, some systems will relocate all active program segments to a contiguous core storage area prior to dynamically loading another module. Other systems only compact storage when core becomes exhausted or when certain high priority programs require loading.

When storage is divided into pages (as described previously in Part I, Section 1.1.2), the program loading function may consist of simply loading all of the required main storage pages, or it may entail preparing the system for a process called paging. Paging requires a fast access secondary storage device (usually a magnetic drum) as a supplement to main storage. Both the secondary storage device and main storage are configured to the same fixed page size. A program resides on both devices; the page of the program containing the instruction sequence currently executing is in main storage, most of the other pages are in secondary storage. Whenever another page is referenced for data or to transfer control, a page area is made ready in main storage—perhaps by moving an in-core page to secondary storage— and the required page is read in. A page map is maintained indicating the physical page assignments (main storage or secondary storage) and the actual storage address (core locations or track). This map is consulted to resolve addresses when accessing the data or instructions within the page. In many systems, this address resolution is accomplished through hardware circuitry; in others, it must be accomplished via software.

Implementation of the paging concept provides for programs of seemingly unlimited size, since a program need no longer be restricted by the actual main storage memory of the computer. On the other hand, paging involves considerable overhead in shuffling pages between main and secondary storage so that the increased capability is not without cost. Because of this cost, several interesting techniques have been developed to minimize the amount of page transfer between secondary and main storage. A common technique is to attempt to determine the page least likely to be referenced in the future and remove this one from main storage whenever a new page area is required. Another technique is to remove only pages that have not been modified in the time since they have been read in. The latter method avoids the requirement for writing the page contents onto secondary storage since the previous version of the page on secondary storage can still be used.

Swapping control – Swapping is a technique of sharing main storage among several jobs by maintaining each job and its status on secondary storage and loading each one into main storage as needed. Though primarily the basis for time-sharing systems, it is also found in non-time-sharing applications under the names roll-out/roll-in and paging.

In a time-sharing system, the swapping mechanism is usually activated each time a program's allocated time quantum expires. The time quantum is the maximum amount of time that a program can control the CPU during each execution cycle. In some systems, the time is fixed either by the system, the operator, or the user. However, many

systems also permit the quantum to vary during execution as the relationship of compute time to I/O processing time or as the relative program activity in relation to other time-shared programs varies. Consequently, time quantum determination is one of the major differences among time-sharing operating systems.

1.1.4 Event Monitoring

Event monitoring refers to the control the operating system maintains over executing programs. The function includes: dispatching control, interrupt processing control, event synchronization, and program limit monitoring.

Dispatching Control - The function at the heart of a multiprogramming or time-sharing system is the dispatching function. Responsible for allocating processor time to each contending task, this is perhaps the simplest of all operating system functional areas in implementation. Simplicity is mandatory because it is one of the most frequently used supervisor routines. If the system is to operate with any degree of efficiency, the dispatching overhead must be held to an absolute minimum. The following description primarily applies to extended multiprogramming systems; however, it illustrates the simplicity of the dispatching function within the most complex type of system.

An extended multiprogramming system must maintain some form of a dispatching queue. This queue contains only those tasks that are capable of using the processing unit. Thus, for example, when a job is scheduled, the program is loaded into main storage, assigned resources, and the program designator is entered onto the dispatching queue. The queue is normally maintained in priority sequence so that the first entry on the queue is the highest priority program available for execution. Consequently, the dispatching routine merely locates the first queue entry, removes it from the queue, and assigns the processor to it. The program is allowed to execute until it is interrupted (in which case the interrupt routine places the program designator back onto the dispatching queue) or until it must suspend itself due to required supervisor service such as I/O processing, etc. When the required service has been performed, the supervisor again enters the program designator onto the dispatching queue. All interrupt processing routines and all supervisor service routines return control upon completion to the dispatching routine to select the highest priority program that can execute next.

In a time-sharing system, the scheduling queue corresponds to the cyclic or round-robin scheduling list and the implementation philosophy is similar. In a basic multiprogramming system or a real-time oriented system, the dispatching queue may in fact be implemented via hardware circuitry rather than software. Nevertheless, the basic concept, as illustrated above, remains the same.

Interrupt Processing - An interrupt, as the name indicates, is a break in the normal flow of program execution. An interrupt is usually caused by a hardware-generated signal such as an I/O event, a program error, a machine error, or an operator-initiated signal.

In a real-time system, the interrupt levels of the various events to be monitored are usually specified when the system is generated; in general purpose systems, they are usually intrinsic to the system and may not be specified. Normally, processing on a given level will be interrupted whenever higher level interrupts occur; interrupts of a lower level than the processing program will be held pending. If several interrupts occur simultaneously, they are stacked and honored according to their priority. Unfortunately, in some systems inadequate stack lengths can cause multiple interrupt signals to be lost. Interrupt acknowledgement can be altered by disabling interrupts to prevent their recognition or by masking them to defer their recognition.

When an unmasked interrupt occurs, the current instruction is usually allowed to complete execution. Then the program and the contents of the machine registers are saved in main storage and control is transferred to a program which will service the interrupt. This program may complete execution or in turn may be interrupted by a still higher priority interrupt. When an interrupt processing program does complete execution, control is returned to the highest pending interrupt processing routine or to the supervisor dispatching routine if no interrupts are pending.

Event Synchronization - Event synchronization is the process of delaying task execution until some specified event occurs or the process of triggering a task upon the occurrence of a specified event. The most common types of events which may delay or initiate task execution are I/O completions, selected time intervals, subtask completions, and unsolicited key-ins.

Event synchronization may also be effected between several tasks of a job. One task may initiate any number of subordinate tasks and may execute concurrently with them. This main task may, at any time, issue wait requests which will suspend main task processing until one or more of the subordinate tasks have been completed.

Program Limit Monitoring - Many systems monitor various resources during job execution to insure that certain specified limits are not exceeded. Limits are usually established for such elements as central processor time used, output records produced, and the amount of main and secondary storage space allocated. While installation maximums for each resource are normally established at system generation time, they can usually be overridden by job control cards. Some systems automatically cause abnormal job termination when any of the system limits for a particular job are exceeded. Other systems permit the user to specify other actions to be exercised, such as operator or program notification, whenever the various limits are exceeded.

In some smaller real-time systems, execution time limits are not explicitly set; however, the supervisor maintains a countdown loop which will time out unless it is periodically reset by the executing program. If the countdown loop times out, an interrupt is generated and program execution terminates.

1.1.5 Program Termination Processing

A program terminates normally when it has completed all of its processing and returns control to the supervisor. A program may also be abnormally terminated by the operating system under a number of different circumstances. This is frequently caused

by a programmed request for abnormal termination of the job, a system determination to abort due to lack of corrective actions for certain error conditions, or a console command to terminate issued by the computer operator. The standard abnormal termination procedure is to discontinue execution of the executing task, or possibly of the entire job, depending upon the criticality of the task with respect to the job.

When a job is terminated (either normally or abnormally), the supervisor deallocates and returns to the system all areas of core and all devices assigned to the job, and performs whatever operator or program notification is necessary. A summary showing CPU time utilization and overhead, local and remote device utilization, file access, and error statistics, etc., may also be recorded. Additionally, when a job is abnormally terminated, some of the following may be recorded: the condition causing termination, a dump of all or selected portions of core, the results of special program-defined abort procedures, and the contents of registers and status indicators.

1.2 I/O Control

System control over the activity of input and output devices is a fundamental feature of third-generation operating systems. This control is maintained for two separate reasons. First, it simplifies the work of the application programmer since he need not be concerned with the intricate details of programming for a variety of channel and device characteristics. This upgrades the overall effectiveness of the programming staff since a standard and well defined approach, rather than a number of widely varying approaches, is always used to interface with I/O devices. Secondly, since the system is in control of I/O activity, the application program need not be alerted to process I/O interrupts. Consequently, in multiprogramming and time-sharing environments, the job management function is considerably simplified by maintaining a single system I/O interrupt processing routine. Thus, the queuing, dispatching, and timing requirements that would arise if multiple I/O handlers were competing for CPU use simply do not exist.

The I/O control function usually supports all devices the hardware vendor supplies with the computer system. These normally include direct access storage devices, magnetic tape drives, paper tape drives, unit record devices, typewriters, remote terminal display devices, and plotters. Additional devices which are uniquely attached to a computer system are not normally supported and require special augmentation programs to bring these devices under supervisor control.

1.2.1 I/O Scheduling

I/O scheduling is the process of acknowledging a request for I/O services and initiating the physical input or output operations to satisfy the request. Requests for service may be processed immediately or they may be serviced according to a queuing scheme. In the latter case, queues are provided to hold requests for channel or device services.

Typically, when an I/O request is issued, both the channel and the device are checked for availability, and if both are free, the operation is initiated. If either is busy, the request is placed on a channel or device queue. Whenever the channel becomes free, the channel queue is checked for pending operations. A pending operation will be scheduled if the referenced device is free; otherwise, the queue will be checked

further for other pending operations on available devices. Some systems also allow a device to be attached to more than one channel. Then, if one channel is unavailable for the I/O operation against the device, the channel scheduler will attempt to use the second channel before entering the request on the channel queue.

Queues are usually maintained in a priority order. This order may be device priority, job priority, or simply first in/first out. In any case, an I/O operation will ordinarily be dispatched according to its position within the queue. However, in some cases the supervisor may consider hardware optimization over priority when selecting an entry from the queue. For example, requests for access to a disk device are frequently serviced in a manner that minimizes disk arm movement, rather than queue priority.

1.2.2 Data Transfer

Data transfer controls the movement of input or output data between main storage and secondary storage, or between main storage and input/output devices. Prior to initiating the data transfer operation, an area of main storage (called a buffer) must be set aside. The buffer will either contain the output data to be transmitted or will receive the input data as it is being transmitted. Techniques for allocating buffer areas vary greatly among the various operating systems.

Many systems, particularly smaller systems, require that the application programmer suballocate parts of his allocated main storage as buffer areas for his files. There are three basic methods that he may use to provide these areas: single buffering, double buffering, and buffer pooling. Single or double buffering techniques allocate either one or two fixed core storage areas as buffers for each designated file. The areas are allocated in advance and remain allocated whether or not the file is being processed. A buffer pool, on the other hand, is a large core storage area containing several buffer areas. Each of these areas can be made available as an input or output buffer for any file. Whenever an I/O operation is required for a file, the user removes a buffer from the pool and assigns it to the file. When the data transfer and subsequent processing are complete, the buffer is returned to the pool. As a consequence, buffer pooling can reduce the total area required for user buffers if all files are not currently active.

More sophisticated operating systems provide system-controlled exchange or dynamic buffering facilities. Dynamic buffering is a technique which provides buffers to an executing task in response to an actual demand for a buffer area. Dynamic buffering is similar to buffer pooling except that the system controls buffer allocation rather than the user. Dynamic buffering is an example of the dynamic core allocation function discussed in Part I, Section 1.1.2.

Exchange buffering is a technique which eliminates the need for moving data between a system buffer and user storage. In this type of processing a system buffer is filled by an input action on a file. Rather than moving this data to a user processing area, the system simply exchanges buffer areas with the application program. Similarly, on output, when a user buffer is full, the system exchanges it for an empty buffer and user processing continues. Exchange buffering allows complete system control of buffer activity while minimizing system overhead.

As data transfer operations occur, the system may have to initiate routines to translate data codes. This occurs frequently in teleprocessing where the input or output line data coding structures differ from the internal computer data codes. When code translation is required, it occurs as a part of the system interface to the I/O device. Thus, differences in coding structures are generally transparent to an applications programmer. Many smaller systems, however, do not provide this feature and the function must be performed by the application program.

1.2.3 Device Manipulation

Device manipulation is a control function which allows a physical I/O device to be positioned without actually requiring data transfer. Device manipulation facilities normally permit volume positioning (rewinding, forward spacing, back-spacing, disk arm positioning, etc.), printer spacing and forms control, and card stacker selection. These features are generally available on every type of operating system for the devices associated with the system.

1.2.4 Remote Terminal Support

Control over remote terminal operations is found in all time-sharing systems and in batch processing systems that provide remote job entry. While it may be possible to attach a remote terminal to practically any computer system, many operating systems are not designed to specifically support remote terminals and special terminal support routines must be designed to augment the normal supervisor facilities.

Operating systems that do support remote terminals as independent input or output devices generally provide a line-service software package to supplement the normal system data transfer routines. There is a great variation in the support provided by these packages. Some systems only provide for a standard terminal processing speed and format while others offer a mix of line speeds in addition to such features as CRT paging and data compression.

In a time-sharing or remote job entry environment, the remote terminal is the main input and output device of the system. Consequently, the system customarily provides all the services of input and output stream control, as described in Part I, Section 1.3.3, to the terminal. In these environments, a capability is usually also provided for the remote terminal user to communicate directly with the computer operator via the terminal keyboard.

Many systems supporting terminals also provide a capability for the various terminal users to send messages to one another. Thus, a user in one location can use the terminal to communicate with users in other localities. Messages can be sent to a particular user or broadcast to all or to a selected list of users. When this feature is available, terminal users may inhibit message receipt by requesting the system not to forward messages from other users. Consequently, this system function may be looked upon as a telephone switchboard controlling a series of separate conversations.

1.3 System Communication

System communication incorporates all of the functions involved in the exchange of information between the user or the computer operator and the operating system. The communication may be oriented either to controlling the execution of scheduled jobs within the system or to configuring system components and monitoring system status.

1.3.1 System Startup

System startup is performed by the computer operator to initialize the operating system for normal processing. In batch processing environments this is the normal beginning-of-the-day procedure once computer power has been turned on. In real-time environments operating around the clock, however, startup is only performed when the system has been shut down for some reason.

System initialization is usually achieved by loading a system tailoring routine. The tailoring routine reads and processes specific system configuration information from the operator's console or the system card reader. The amount and type of configuration information supplied is dependent upon the capabilities of the total system and the versatility of the operating system. Options provided by various systems include patching the supervisor, specifying partition sizes, specifying system resource availability, and setting the system clock. A few systems (for example, General Electric's GECOS III) allow a complete form of system tailoring tantamount to system generation (see Part II, Section 1.0) during each initialization.

When teleprocessing capabilities are present, the supervisor may also incorporate a special start-up routine to initialize the system for communications processing. This is frequently required to build communication buffers on secondary storage, open the on-line processing files, and establish the teleprocessing line connections.

When all of the startup functions have been performed, the system is ready to initiate processing. In some systems, control is passed directly to input symbionts which will read the input job control streams and develop the scheduling queues. In others, the system enters an idle state and remains there until an interrupt signals the initiation of a real-time or time-sharing processing requirement. In this case, the operator may also issue an interrupt to initiate batch job processing.

System restarting is required when a failure that affects the total system, rather than a specific job, occurs. Restarting functions are oriented to restoring as much as possible of the system environment that was valid at the time of the error. In critical real-time environments, for example, system checkpoints are frequently taken at regular intervals. These checkpoints can be used to reload a previous valid version of the operating environment when no other immediate method of repair is possible.

Recovery from certain hardware malfunctions may indicate a need to replace certain supervisor routines that have been damaged. The use of refreshable modules greatly facilitates this procedure. A refreshable module is one that is not modified by itself or by any other program during execution. If there are indications that program damage has occurred, and if the damaged module is a refreshable module, the system merely replaces it with a new copy.

Other restarting elements may also be provided to dynamically reconfigure the system by eliminating failing components such as I/O devices or storage elements. In general, the more critical the environment, the greater the capability for system-controlled restarting. Real-time systems, for example, are frequently designed primarily around the restart capability.

In batch processing environments, restarting may be designed to provide recovery for incomplete jobs, re-initiation of interrupted system output, and preservation of in-core accounting information. Extended multiprogrammed systems generally retain the scheduling queues of all jobs not executing at the time of system failure though they may or may not be able to recover jobs in execution. When no total system restart facilities exist, a system restart is identical to an initial startup.

1.3.2 Job Control Communication

Job control communication refers to that communication between the operating system supervisor and either the user or the computer operator relating to the initiation, running, or termination of individual jobs within the system. In batch processing systems, user/system communication is generally non-interactive, whereas in time-sharing systems it is almost always interactive. Operator/system communication is predominantly interactive.

Non-interactive user communication frequently takes the form of system control cards imbedded within the input job stream indicating the hardware requirements for a job (I/O devices, files, main storage, time, etc.), the sequence of task execution, and pertinent error procedures. Frequently, these specifications can be stored within the system as a cataloged procedure and invoked by the user on a single control card. Interactive user communication consists of single commands issued to the system from an on-line terminal as the job is executing. Interactive commands may also invoke pre-defined sequences of commands.

Operator communication is usually conducted via the operator's on-line console. Generally, the operator can control the actual execution of a job by raising or lowering its priority, by assigning or removing I/O devices, and by suspending or cancelling a job when necessary.

1.3.3 Input/Output Stream Control

The input job stream is the sequence of system control statements and program data submitted to the operating system on an input device specified for this purpose. In serial processing and basic multiprogramming systems, the device tends to be the system card reader, though, in fact, any input device can be used. In these two system types jobs are read, processed, and output in the order in which they occur in the input stream.

Input stream control consists not only of reading the input stream, but of recognizing the presence of system job control cards within the input stream and transferring control to the proper supervisor routine for interpretation and processing. Program data that appears in the input stream must be distinguishable from system control cards. Special system control card identifiers may be used or delimiter cards may be inserted

to precede and follow the data cards. In either case the input stream control function must not permit an application program to read beyond the program data included within the job stream.

The output stream consists of diagnostic messages and other data issued by the operating system or the processing program to the output device specifically designated for user communication. In practice, the output device is normally the system printer; though, again, any device could be chosen.

In larger systems, particularly extended multiprogramming systems, the input and output streams are maintained as separate files in direct access storage. Programs called symbionts are used to read and transfer the system control and data cards from input devices to the input stream files. Other symbionts transfer output data from output stream files to the actual output devices.

The advantage of intermediate input and output stream files can be best illustrated with an example. Consider the concurrent (either multiprogrammed or time-shared) execution of two independent application programs in a system that has a single system printer. If both programs require the use of the printer, only one can physically be assigned the device. If the device were assigned to both programs, output data from both jobs would appear intermixed in the listing. However, if one program is assigned the device, the other must be suspended until the device again becomes available. On the other hand, if both programs create separate output stream files on a direct access device, both programs can execute concurrently. When each program closes its output stream file, the file can be transferred to the printer by an output symbiont when the printer is available. Thus, the single system printer does not inhibit concurrent processing. A further benefit is that the system printer is not reserved during the entire execution period of either application program. Rather, it is reserved only for the length of time it takes to transfer the output data from secondary storage.

Thus, symbiont processing enables input/output devices to be utilized at physical data transfer rates, while permitting programs to process input and output stream data at storage transfer speeds rather than at the lower peripheral device speeds. The overall effect on the system is a considerable increase in the throughput without requiring additional peripheral devices.

Since, in time-sharing applications, the terminal is usually dedicated to a particular time-sharing job, and since both the input and output stream are usually located at the same terminal, no significant equipment or time saving is afforded time-sharing programs by using symbiont processing techniques. On the other hand, when the terminal is used for remote batch processing, symbiont processing can offer both time and equipment savings, particularly when multiple jobs are submitted.

1.3.4 Resource Status Modification

In most computer operating environments, it is desirable to alter the computer configuration without physically terminating all system operations. For example, a tape drive may require cleaning, a disk file may require maintenance, or an off-line

operation may have concluded and additional peripheral devices may have become available for system use. As a result, most systems allow the computer operator to alter the status of resources available to the system during system operation. Generally, this is accomplished via direct operator console commands to the operating system supervisor.

The resources affected may vary from peripheral device utilization to main storage and CPU allocation. The following list is indicative of the types of modification provided in several systems: addition, deletion, and replacement of input and output devices; modification of partition sizes and/or their relative priority within the multiprogramming scheme; initiation or termination of activity in any or all partitions; and alteration of the amount of CPU time allocated to various processing types supported by the system.

1.3.5 System Status Interrogation

The computer operator is usually given the capability of displaying the status of various system elements. This may range from the status of specific jobs to the status of I/O devices and main and secondary storage. Systems vary considerably in the capabilities provided and where some may only provide status information on particular items, other will produce extensive visual displays showing the current status, relative usage, and accumulated error statistics for any system element.

In time-sharing systems, each terminal user can usually query the system on the current status of his job, the current state of his and other terminals, and the current status of system I/O devices.

1.4 Recovery Processing

Recovery processing is invoked whenever an external error (as opposed to a programming error) occurs that prevents a particular job from continuing normal processing. The recovery routines attempt to avoid abnormal job termination by restarting the executing job from an earlier point in the processing cycle. Although recovery processing may also be required for the entire system, a discussion of this situation was presented earlier in Section 1.3.1. The following discussion is therefore concerned only with job, not system, recovery.

1.4.1 Checkpointing

Checkpointing is the recording of information about a program and its environment on secondary storage so that at any future time the program may be re-initiated from that point. Small operating systems, particularly serial processing systems, checkpoint all of core while most multiprogrammed systems checkpoint only individual storage partitions. The checkpoint record is maintained on an on-line (usually tape or direct access) checkpoint file and usually contains main core storage contents. In addition, checkpoints may also contain selected register settings, I/O device repositioning requirements, and the contents of critical permanent and temporary data files.

A checkpoint record is created whenever a request for a checkpoint is issued. The request may be initiated by the operating system, by a problem program, or by a computer operator command. In general, the operating system initiates checkpoints to accomplish roll-out/roll-in processing or for automatic restarts in real-time environments.

User programs usually initiate checkpoints only when system-directed checkpointing is not provided. The computer operator normally initiates a checkpoint only when he must temporarily terminate system operations for some reason.

Some systems only support checkpointing for certain types of jobs. For example, IBM 360 DOS permits background but not foreground checkpoints. Many time-sharing systems do not even provide a specific checkpoint capability. However, some of these systems do allow the user to save the current status of his program for subsequent reloading. This could be considered a de facto checkpoint, since the program can be re-initiated from that point.

1.4.2 Restarting

Restarting is the process of restoring the status of a job to some previous point in time. The three basic types of restart capabilities used within all operating systems are checkpoint, task, and job. A restart taken from a checkpoint re-establishes the program and its data in the operating environment as they existed when the checkpoint was taken and resumes execution at the restart address included in the checkpoint. A task restart re-initiates processing from the beginning of the identified task. A job restart re-initiates processing from the beginning of the entire job.

Except in some cases where the supervisor checkpoints a program to provide roll-out/roll-in services, restarts normally require device repositioning. The extent to which devices may be repositioned varies, although most systems will, at a minimum, reposition sequential devices. Some systems also reposition direct access device arms and re-establish broken teleprocessing line communications. Repositioning not performed by the system must be provided by the application program or the computer operator.

2.0 DIAGNOSTIC ERROR PROCESSING

The diagnostic error processing function is responsible for recognizing hardware, program and interface errors. Recognition is usually based upon hardware interrupts or program testable switches. The function also supports the diagnosis and resolution of error conditions.

2.1 Hardware Error Control

Hardware error control encompasses the recognition of hardware errors and the corresponding support provided for error recovery. Error control routines are generally available for: core storage parity errors, processor errors, I/O control memory parity errors, I/O data parity errors, and power failures. Although recovery routines are usually provided with the system, they may be augmented by the installation during system generation. As mentioned previously, real-time processing environments are particularly concerned with error recovery procedures and often the entire vendor-supplied error recovery system is re-designed for these environments.

When an error occurs, control is transferred to an appropriate diagnostic routine to determine the cause of the error and possibly the extent of the damage. If the error is minor, as in an I/O data check, the operation is reattempted to determine whether or not

the error is permanent or merely transient. Permanent errors usually require some form of operator-controlled action, such as replacing the failing element. If the error is transient, a tally may be updated to indicate which system elements are producing a relatively high level of transient errors.

Major errors on non-real-time systems normally require termination of system operations and some form of corrective maintenance. If the system has extensive job recovery routines as described earlier, operational processing of the jobs may be resumed once the maintenance has been completed. If not, a system initialization and startup will most likely be required.

In order to minimize the likelihood of error occurrence, some systems provide on-line diagnostic testing of system hardware components. The testing routines are comparable to the off-line tests run during scheduled system preventative maintenance, except that they are executed within a normal operating environment and do not disrupt other concurrent system processing. These routines may be scheduled to execute on a periodic basis by using the system's time-initiated scheduling capability or they may be explicitly invoked by the computer operator or hardware engineer.

Some systems, particularly real-time systems, also provide dynamic reconfiguration capabilities for hardware errors that affect overall system performance. These capabilities remove failing hardware components and replace them with alternate or backup devices. If a backup device is not available the system can "fail soft", which means that it will continue to perform with less than the normal number of hardware components. Generally, when a system fails soft, the execution work load is reduced, as necessary, by not processing lower priority programs.

2.2 Program Error Control

Almost all systems recognize program errors occurring in user programs and assume control to prevent the system from being adversely affected. The user is frequently allowed to supply his own error handling routines for certain types of errors, such as arithmetic and data errors. More serious errors, from a system integrity standpoint, are core storage violations, invalid addresses, and improper use of privileged instructions. The occurrence of these errors will usually result in abnormal termination of the offending program.

A different approach to program error processing is taken by most time-sharing systems. Since the terminal user is interacting with the system, an elaborate error processing sequence is not generally required. Instead, the system can simply notify the terminal user of the program error, and allow him to determine what corrective action, if any, should be taken.

2.3 Interface Error Control

Most systems edit interface message formats with the operating system prior to acting upon a communicated request for system service. For example, input stream control commands, operator key-ins, remote terminal communications, and program linkages to

the supervisor are invariably edited in great detail for accurate formats, calling sequences, and valid codes and parameters. Errors will normally result in job termination, task termination, command rejection, or a request for operator or user clarification.

3.0 PROCESSING SUPPORT

The processing support function consists of supervisor routines which may be called upon to accomplish a variety of miscellaneous services for an application program. In general the services are utilitarian in nature and provide convenient, rather than necessary, functional support. The services are described below in terms of timing, testing and debugging, logging and accounting, and system description maintenance facilities.

3.1 Timing Service

Most systems have an internal timing device which provides timing services to application programs. A few systems, to reduce the size of the resident supervisor, only include these services as an option during system generation.

Timing services tend to be of two types: interval timing and real-time clock services. Interval timing services are used to suspend processing for specified times or to alert the program when a specified time interval has passed. Interval timing services may thus be used to schedule intermittent checks for specified conditions or to serve as watch-dog timers to prevent unending program loops.

Real-time clock services are most frequently used to provide the date and time of day to executing programs. However, they may also be used to schedule interrupts at specific times (rather than intervals) so that events can be scheduled or terminated on a time-of-day basis.

3.2 Testing and Debugging Service

The facilities available for testing and debugging application programs vary considerably among operating systems. Some systems, such as IBM's System 360/OS, provide an independent program testing and debugging package that operates in conjunction with the resident operating system. Other systems, such as Honeywell's Mod 4 OS, incorporate testing and debugging support as a part of the resident system service package. Thus, not only do the services offered vary, but the method of implementation may also vary from system to system.

Many systems maintain two distinct user operational modes: a normal mode and a testing mode. Whenever errors occur under a normal mode of program operation, the program is usually subject to abnormal termination. However, under a testing mode, the occurrence of errors usually triggers a series of prescribed debugging routines which gather extensive information about the error condition. For example, storage dumps and selected data file dumps are frequently requested for programs abnormally terminating in a test mode. The advantage of a test mode is that the routines are not invoked until errors occur, whereas in a normal operating mode, or in a system where no distinction is made, their invocation is usually unconditional.

Storage dumps generally allow the user to display all of core or only specified portions. They are frequently initiated during the execution of a program (snapshot dumps) as well as upon termination of the program (post mortem dumps). Generally, these dumps are highly formatted and are oriented to providing as much information about program status as can be presented.

Tracing facilities provide a sequential history of program execution. In general, the history records each particular instruction being executed, its address, the data fields affected, and their contents before and after the operation. Though there are many different types of traces, they all generally display the same types of information and are only distinguished by the occurrences that cause the information to be displayed. For example, several common types of traces are: full instruction tracing, instruction tracing within specified address limits, traces of interrupt occurrences, supervisor entry and exit sequences, and traces of instructions which modify selected words in main storage.

In an interactive time-sharing environment, testing and debugging facilities usually are quite extensive. In this environment the terminal user can normally examine, and modify task elements such as instructions, numeric values and coded information, insert breakpoints into a task, and control execution by directing or redefining the instruction executing sequence.

3.3 Logging and Accounting

Operating systems normally provide accounting and statistical information on job execution and resource utilization. In most systems this function simply records the following information at the termination of each job: job identification and termination status, number of records added to or deleted from each permanent file, CPU time utilized, time used by each channel and each device, number of lines printed, number of cards punched, and number of records written on system output units.

While most systems record this information, it is generally left to the installation to write the accounting and analysis routines required to process the statistical summaries, although a few systems do provide system programs which can process this type of information.

Many systems also provide error statistic accumulation to identify hardware devices or programmers that have a greater than normal occurrence of error conditions. It is assumed that some form of repair or replacement would be undertaken in either circumstance.

Some systems also provide logout facilities which record the environment of the system during a system failure (CPU malfunction, power outage, etc.) so that a post mortem analysis may be conducted by software and hardware engineers.

3.4 Program Accessible System Description Maintenance

Almost all systems maintain a certain amount of descriptive information in a supervisor communication region that may be interrogated by an application program. Three types of information are likely to be maintained: system defining information, current system status information, and individual job information.

The hardware configuration, operating system description, and values of system limits are examples of the types of system defining information which may be maintained. The system status information may include current device status and allocations; a list of currently executing jobs, interactive users, and active terminals; and the limits of allocated core storage. While the information provided in both categories is fairly sketchy in most systems, a general purpose application program or compiler can modify itself to conform to the implemented hardware and software characteristics of the particular installation by interrogating this information.

The individual job information generally includes the name, account number, core limits, and elapsed time and processing time used by each active job. Though it is perhaps questionable as to how the application program might utilize this information, very little overhead is involved in making the information available since it must be maintained for normal system accounting purposes.

1.0 OPERATING SYSTEM MANAGEMENT

Operating system management routines are provided by all systems to enable each installation to generate and maintain a version of the computer operating system.

1.1 System Generation

System generation is the process of tailoring and adapting a generalized operating system to the specific machine configuration and operational requirements of an installation. The generalized master operating system is normally provided by the vendor to every installation. The master system contains all the operating system routines required for any allowable system configuration as well as the vendor-developed optional routines that might be desired by a particular installation.

The master system also contains a series of executable programs which will initialize the system components for the system generation process and create the specific installation version of the operating system. In the larger systems this initialization package is transcribed to a mass storage medium; in the smaller systems, it may be bootstrapped directly into core. The initialization package usually contains a basic executive program, an assembler, a loader, a linkage editor, and special routines necessary for implementing the system.

The actual system generation process consists of generating an operating system supervisor based upon specifications supplied externally -- usually through card or console control statements. This may be viewed as a multi-step process. First, the installation's operating configuration is determined by interpreting specifications for the size of main memory, the number and address limits of each partition, the number of CPU's, the number of I/O controllers or processors, the number of communication controllers or processors, the type and designation of each I/O device, and the type and designation of each local and remote terminal. Additionally, this step usually includes differentiation of device roles such as distinguishing between user and operator terminal devices.

Next, installation tailoring criteria for system software features are interpreted and the features are incorporated into the generated system. For example, the establishment of memory protection limits, error recovery options, and accounting options would occur within this step. Many of the tailoring capabilities are highly dependent upon the type of operating system orientation that the generalized system can support. For operating systems supporting real-time processing, specifications for the number and type of interrupt levels, their respective priorities, and processing assignments are normally specifiable. Batch processing systems generally allow scheduling and dispatching priorities to be defined, while time-sharing systems permit authorized user declarations, passwords, accounting controls, resource limitations, and priorities to be enumerated.

Several systems also permit specifications of those operating systems supervisory modules that are to be core resident and those that are to be transient. The use of this feature is extremely desirable in those situations where the usage of system modules may vary greatly from installation to installation. By including frequently used routines as a part of the resident system, operating overhead can be significantly reduced.

The last step involves selecting from the wide range of operating system support programs, those which will be included within the system to satisfy the specific application requirements of the installation. By including only the appropriate compilers, data management systems, and utility programs needed by the installation, the overall size of the operating system file may be kept to a manageable level. Most systems also provide facilities for incorporating user-developed routines during this final step to augment the operating system support programs.

The resulting generated system is capable of supporting normal installation processing. In most cases, the system file is maintained on an on-line immediate access device. When immediate access devices are not available, however, the file is normally maintained on magnetic tape. Unfortunately, tape systems, due to their serial orientation, tend to have a much higher overhead rate than the other systems.

1.2 System Maintenance

System maintenance allows an installation to update the operating system in response to changes in the operating environment or to changes of the programs within the operating system itself. The latter category is generally related to vendor-supplied updates which are distributed periodically. Consequently, this type of maintenance tends to be performed fairly regularly and generally necessitates suspension of all other processing activity until the update is complete. Extensive updates to the system may involve a total regeneration of the system; however, minor changes may frequently be effected by simply replacing selected system modules.

Dynamic maintenance is a capability whereby a system can alter its physical configuration tables on-line to adjust to additional or deleted processing elements. This capability is particularly important for critical real-time systems. As described in Part I, Section 2.1, some systems also have the capability to fall back into a degraded state of operation (fail soft) upon the failure of some processing elements which cannot be replaced by others.

Some systems, such as IBM's TSS/360, also allow the individual terminal user a limited system tailoring facility. These systems allow the programmer to adapt certain features of the system to his particular requirements without disrupting the utilization of the same features by other users. The terminal user thus creates his own system profile which is activated only when he properly identifies himself. Each user may maintain an entirely distinct profile without affecting other users.

A few systems provide an on-line testing capability for updates or modifications to the operating system. If an error occurs within the system module being tested, the system will restore the unmodified version of the module. Consequently, modifications can be tested without requiring regeneration of the entire system; further, the testing can be conducted in a live environment rather than in a special off-line mode. This approach to system checkout is most valuable in two environments: first, where there is such a diversity of system use (e.g., a time-sharing interactive and remote batch environment) that off-line tests could not exhaustively test the modified module; and second, where continuity of operations is critical (e.g., in some real-time environments).

2.0 PROGRAM MAINTENANCE

Program maintenance facilities support the maintenance and modification of application programs within the framework of the system. These facilities are distinct from the support features that an application program can call upon when executing; rather these facilities treat the application program as a product by itself.

2.1 Library and Directory Maintenance

Many systems provide capabilities for maintaining one or more program types in indexed libraries. In general, a program library is a collection of routines or instruction sets which are all represented at the same code level. The code levels generally found in contemporary operating system libraries are: macro code, source program code, relocatable program code, executable code, and job control procedures.

Macro libraries are composed of unique source code instruction sets which perform relatively small and distinct functions. When writing a program, a programmer may invoke a reference to this library by inserting a macro library call statement within the program source code statements. The language translator will recognize the call statement and replace it with the appropriate instruction sequence from the macro library during compilation.

Source program libraries contain routines or programs in the original programming language used by the programmer, such as COBOL, FORTRAN, etc. Source programs may be combined with other routines for compilation or input directly to an assembler or compiler.

Relocatable program libraries contain the relocatable programs which are produced by the system compilers and assemblers. Generally speaking, a relocatable program is not directly executable and must be converted into a load module prior to being loaded and executed. Libraries of common relocatable programs are useful when a single routine, such as an input/output routine for a transmission line, might be used by several programs. Retrieving it from the relocatable library saves recompiling the routine for each separate program requiring it.

Load module libraries consist of programs in executable form which may be directly loaded into main storage and executed. Several systems make a distinction between user and system load module libraries. The former consist of various application programs whereas the latter consist of the executable programs comprising the operating system proper. The advantage of maintaining separate libraries is that a user library may be taken off-line when it is not required.

Systems which require a great number of system control cards for the execution of a job will frequently provide a procedure library to contain defined system job control card sets. A single control card in the input stream may then be used to reference such a procedure in the library. The system will in turn process the job control card set from the procedure library as if it had been in the input stream. This relieves the programmer of the responsibility of preparing an elaborate set of job control cards each time a job is submitted.

Various utility functions are provided for the maintenance and modification of these libraries. The primary facility is the cataloging function which enters new elements into the appropriate library and updates the library directory. A system may provide both static and dynamic cataloging capabilities. Static cataloging requires the explicit execution of a library maintenance program to perform the cataloging function. Dynamic cataloging permits the user to add an element to a library without explicitly invoking a librarian program. For example, a user may request the system to catalog a job or procedure upon successful execution.

Other library and directory maintenance facilities include copying, renaming, punching, listing, and displaying library elements as well as allocating or deallocating library space on secondary storage devices.

2.2 Load Module Generation

A load module is an instruction set that may be directly loaded into the main storage of the computer and executed. An absolute load module can only be loaded into one specific main storage location for execution, whereas a relocatable load module may be loaded into any main storage location. Most contemporary systems do not directly produce executable load module code from either the system compilers or assemblers. Rather they produce what is termed relocatable code (which should not be confused with relocatable load modules). Relocatable code is an instruction set coded in machine language but which for one reason or another is not directly executable. For example, it may not have been assigned physical main storage locations from which to execute, or it may have unresolved linkage sequences to the supervisor or to other instruction sets. The process of converting these programs to a format capable of being loaded and executed is called load module generation.

The element of the system that performs this function occurs in two different forms depending upon the system. In some systems, it is an independent job step that must be scheduled explicitly; in others, it is a supervisor service that may be dynamically invoked by an executing task. Further, the resulting load module may, in some systems, be immediately executed, while in others it may require subsequent loading as part of a separate job step.

In many systems, capabilities exist to combine the relocatable code modules produced by various language compilers with subroutines and other relocatable code modules to provide a single load module. When these modules are combined, they are normally assigned to contiguous storage locations and the internal storage references within each module are appropriately modified. Next, the inter-module linkages (whereby one object module transfers control and data to another) are resolved by determining the newly assigned storage locations of the respective modules and inserting these addresses into the linkage sequence. This process is known by a number of different names; the more frequently used terms are binding, linkage editing, and collecting.

Many systems also offer an additional capability that provides for the implicit inclusion of relocatable elements. If any inter-module linkages are unresolved at the end of the linkage editing process, the system assumes that the unresolved entries are implicit

calls for relocatable subroutines and initiates a search of the system library for subroutines of the same name. This feature is particularly handy for program code generating routines (such as compilers, data management systems, report generators, etc.) since an explicit list of supporting subroutines need not be specified during code generation.

3.0 COMPILER INTERFACES

Those operating system functions that provide supporting services to the system language compilers are the least standard of all operating system functional categories. Furthermore, differences can not be directly attributed to the size, orientation, or sophistication of the operating system. As a result, only a very few systems provide all of the functions noted below and several do not provide any. Because of this wide diversity, the functions are only described in general terms.

3.1 Executive Routine Support

Many operating systems grant the system compilers selected privileges which are not available to other non-supervisory programs. For example, several systems allow the compiler to use communication tables within the resident executive for passing parameters and storing specifications about the compilation. Many systems also provide special job control cards for compilers which include compilation parameters along with normal operating system parameters. In some cases the executive system may actually decode these parameters and place them in an appropriate compiler communication table.

Other types of executive routine support are found in those systems that recognize uniquely formatted compiler input/output files and provide non-standard input and output symbionts for processing them. Finally, some systems also provide a series of compilation error codes which can be used as conditional scheduling parameters by subsequent tasks and steps within the job.

3.2 Library Support

A few systems provide and maintain unique libraries for the exclusive use of the system compilers. These libraries may take the form of compiler source program libraries, macro statement libraries, or compiler subroutine libraries. In general, the same maintenance facilities that are available for other system libraries are available for compiler-oriented libraries.

3.3 System Utility Program Support

Some compilers have the capability of generating linkage sequences to selected system utility programs in order that these facilities can become available to the compiler language programmer. For example, COBOL compilers commonly allow references to the system sort and merge functions. Other utility functions that can be invoked in some systems are the data management and data handling support functions described below in Part III.

4.0 MANAGEMENT SUPPORT UTILITIES

The management support utilities are primarily for the use of the system manager rather than the average user. Utility functions provided for normal users are described in Part III,

Section 2.0. The functions described in this section enable the system manager to add and maintain sequential and direct access volumes, simulate certain system facilities for testing purposes, obtain statistics of system and device performance, and initiate recovery from total system failures. In many cases, these routines may not directly interface with the system supervisor and may be executed independently.

4.1 Peripheral Device Support

This function is normally invoked when a new volume is to be added as a system component. The volume preparation function writes a standard system label on the volume, formats the records and/or tracks where track formatting is required (such as for disk files), creates any necessary volume directory entries, and allocates space for additional directories, communication buffers, etc. Only upon completion of this function is a volume available for normal system use.

Volume maintenance functions may also provide diagnostic routines to verify the correctness of all volumes in the system. Normally, these routines perform a surface analysis of the volume to detect failing surface areas and identify or replace any defective areas. Certain file purging functions may also be invoked to erase selected areas on a volume or to clear main or secondary storage.

4.2 System Simulation Routines

The capability to simulate certain system functions is of paramount importance during the development of many real-time and teleprocessing oriented programs. For example, if certain real-time programs can only be invoked by the occurrence of specific interrupts, these programs must be tested by providing the actual interrupt. This may be difficult to effect or may disrupt the operational system. Consequently, a capability to simulate the occurrence of the interrupts and to thereby invoke the program permits much greater flexibility in program testing.

Similarly, an exhaustive test of communication and line support programs might require a scenario so complex that it could not be performed from available support terminals. Again, the capability to simulate the arrival of a series of communication messages provides a needed testing capability. Many of these routines might also be used by the system manager to test and validate the operating system itself. Unfortunately, few systems provide extensive capabilities in this area. Those that do are primarily small systems for real-time applications rather than large, general purpose systems.

4.3 System Measurement Routines

Systems measurement routines enable the system manager to obtain various statistics about the operational use of the system. Typical statistics include job throughput times, file or device utilization figures, and the identification of bottleneck conditions. In addition, there are a few systems that provide visual displays of current and past system utilization reflecting error statistics on all configured devices, channel usage, memory allocation, programs in core, programs swapped out of core, and processor time consumed by each program at that point in time.

The proper interpretation of these statistics enables the system manager to better "tune" the operating configuration to the actual processing environment by adding new files or devices, redistributing files between devices, modifying the scheduling and/or dispatching algorithms, and by changing the degree of multiprogramming.

Unfortunately, too few systems provide the necessary statistics to adequately measure system performance. For this reason several independent software vendors provide supplementary programs to support the operating system in the performance of this function.

4.4 Stand-Alone Utilities

These routines are primarily used when the system has failed and normal diagnostic routines are incapable of restarting the system. Two types of routines are provided. The first type includes those status display routines which produce core dumps, file dumps, and dumps of selected diagnostic log-out areas. These dumps are presented for interpretation by a hardware or software engineer to determine the cause of the failure.

The second type consists of recovery support routines which enable the system to reconstruct much of its pre-failure environment and to reinitiate processing by rebuilding selected queues, reloading various programs, and restarting checkpointed programs. The recovery support functions are most often found in systems oriented to supporting real-time or communication-based processing.

Part III: Data Manipulation Functions

1.0 DATA MANAGEMENT

The data management functions consist of file management facilities, I/O support facilities, and data management system facilities. File management facilities provide file support for the system files as entities rather than for the individual data records within the files. Support for the latter is provided by the other two data management categories.

I/O support facilities enable a program to access and process individual data records within the file. These facilities are normally invoked by issuing macro instructions within the problem program and eliminate the need for programmers to be concerned with many of the problems of reading and writing data records.

Data management systems allow a user with limited programming interests to perform certain functions on a data file, such as retrieving and displaying selected portions of the file. Generally, data management systems are adjuncts to an operating system and are more or less self-contained depending upon the architecture of the particular system. Consequently, a data management system will make use of many of the features provided by other operating system functions in its own internal design.

1.1 File Management Facilities

As indicated above, file management facilities are oriented toward controlling the various data files within the system. File management functions are invoked to locate on-line and off-line files, permit or restrict user access to files, and to provide backup and restoration services in case of file damage.

1.1.1 File Location Recognition

In most systems permanent data files are identified by labels assigned by the user or the system. File labels may be composed of such items as the file identifier, the file edition number, the owner's name and account number, and perhaps a file utilization privacy code. Several systems permit file names to be a composite of several names in order to provide hierarchical levels of file indexing.

In larger batch processing and time-sharing systems the location of all permanent data files known to the system is usually maintained in a master directory or catalog. A file can then be located for processing by searching the master directory of on-line and off-line files. However, if the system does not maintain a directory, a sequential label comparison must be performed physically against each on-line file until the desired file is located. In this environment, there is also usually no capability to locate off-line files; instead they must be presented to the system by the operator.

Whenever master directories or catalogs are maintained, various cataloging controls are available to the user. In general, he may add new file descriptions to the catalog and modify or delete existing descriptions. He may also be allowed to extend or contract the amount of space made available to an existing file or to alter the list of

authorized file users. Additionally, all of the features available for maintaining system library directories (as described in Part II, Section 2.1) are usually available for file directories as well.

1.1.2 File Access Control

The designation and restriction of file access may be a function under control of the operating system or it may merely be established by a set of installation programming conventions. When controlled by the system, the owner of a file can usually designate that the file is to be maintained for his use only, for the use of a designated group only, or for general use. Frequently, the owner may also specify the level of access afforded each designated user. For example, access may be restricted to a read-only level for some users while others are allowed full read and write capabilities.

Concurrent access control is required to maintain the scheduling and handling of concurrent or simultaneous requests for a data file from separate programs or users in a multiprogramming or time-sharing environment. Basically, the control function must determine if multiple user access can be permitted or whether the file must be restricted to single user access. In situations where multiple users may simultaneously access a single file, it is usually desirable to grant any number of them read-only access, but to restrict write access to a single user at a time.

1.1.3 Backup and Restoration Facilities

Most file-oriented systems require a capability to recover from inadvertent damage to the file system. Consequently, some operating systems provide facilities for maintaining various types of backup files: checkpoint files, transaction data files, and previous versions of the data file (grandfather files). These files may be available to the system on either an on-line or off-line basis. Restoration functions may be initiated automatically by the system (particularly in real-time systems) or may require operator intercession.

1.2 I/O Support Facilities

An important distinction is made in describing the levels of I/O support provided by a system. The basic level of support, physical input/output, is provided by routines that initiate the actual data transmission process and provide program access to the data in the format of transmission. This function includes most of the features discussed in Part I, Section 1.2.

The extended level of support, logical input/output, allows manipulation of data without regard to the physical structure of the data. It thus serves as an intermediary between user data handling operations and the physical input/output services of the system. All systems provide physical input/output support and all but the smallest systems provide logical input/output support.

1.2.1 Data Access Control

The most common file access methods supported by contemporary operating systems are the sequential, indexed, keyed, random and teleprocessing organizations. Each of these access methods can provide for data storage and retrieval on a physical or logical record level.

Sequential access methods process records serially and read or write them consecutively on a storage volume. Sequential access may be provided for data stored on any secondary storage medium; although, certain storage media, such as magnetic tape, obviously dictate a file organization of this type.

Indexed access methods create and maintain files which, in addition to the data records, have directories of selected record field values and their corresponding record addresses. Records may then be located by searching the directory rather than the file. Some form of immediate access storage, such as disk, is necessary for indexed access methods to have value.

Keyed access methods rely on a selected data field within each record to uniquely identify the record. This data field, called the record key, frequently corresponds to the record identification code, though it need not do so. Keyed access is useful for secondary storage devices which have a physical design that features hardware-implemented search instructions. In such systems, a record request will cause the read/write mechanism of the storage medium to scan the entire file in search of a selected record key. The technique is advantageous since processor time need not be spent in scanning secondary storage and may thus be employed with other execution functions. The technique is also frequently augmented by software which isolates a portion of the file prior to issuing the keyed search in order to avoid a scan of the entire file.

Random access methods read and write records without regard to the physical sequence in which they are stored. Consequently, records stored in this type of organization must have some type of identification code that will enable the record location to be determined. Usually, an algorithm is used to convert a unique record identification into a unique storage address on an immediate access storage device.

Teleprocessing data is usually composed of character strings of varying lengths. While not a file in the classical definition, most systems provide assistance in accessing and processing the relatively unformatted messages that accumulate in the system communication buffers. This assistance normally handles all communication between the computing system and remotely located terminals. The functions performed include allocation of storage for message buffering, polling terminals to determine if any have messages to send, recognizing message priorities, analyzing message headers to determine where input and output messages are to be routed, checking the sequence numbers of incoming messages, checking transmission errors, and maintaining input and output message queues.

1.2.2 Data Blocking/Deblocking Control

Blocking combines two or more individual records into one physical data block; deblocking isolates the individual records within a physical data block. Record lengths may be fixed, variable, or undefined and all of these types may be blocked or unblocked.

Operating systems which only provide physical input/output support require the user to perform his own record blocking and deblocking functions. Operating systems which provide logical input/output support allow the user to operate on all data at the individual record level without regard to the structure of the physical block. Blocking and deblocking are usually accomplished by moving data between the input/output buffers and a user processing area or by using special location pointers to isolate and process information within the buffer.

1.2.3 Label Processing

Most systems provide facilities for writing and checking file labels when a data file is opened or closed. Many systems also permit the user to specify his own labels and to supply special routines for processing them. A few of the smaller systems provide no label generation and checking facilities, and all label processing functions must be performed by the user.

1.3 Data Management System Facilities

A data management system is a group of integrated routines developed to create and maintain an organized collection of related data, known as the data base, and to interrogate the data base and produce various types of formatted reports. A data management system will create files from various input sources; maintain these files by additions, deletions, and alterations; create new files and reorganize and merge existing files; select data via user-prepared queries; make computations on this data; and produce reports in system-defined or user-specified formats. A data management system may be designed to operate in either a batch or interactive mode.

1.3.1 Control Specification

A data management system must be provided with a set of specifications or commands delineating the jobs it is to perform. The system interprets these specifications, and generates functional modules to perform the selected jobs. These modules may take the form of interpretive tables or executable programs or subroutines. Generally, the system will maintain an extensive library of functional subroutines which may be incorporated as needed into the final support modules. These subroutines range from arithmetic and data conversion routines to file searching and positioning capabilities. The generation of the resulting support modules is analagous to the process used by a compiler to convert user code into machine executable code.

1.3.2 Data File Generation and Maintenance

Data file generation and maintenance is concerned with defining the internal structure of a file, allocating space for the file on a storage device, processing input transactions against the file, performing logical and interactive maintenance on the file, and reorganizing the file when the structure must be modified.

Internal Structure Definition - The most common file structures are sequential, hierarchical, indexed, list, and ring. A sequential structure is one in which the data elements are all of equal rank and are maintained in a serial order. In a hierarchical structure the data elements are classified and stored according to a ranking scheme. An indexed structure is one in which portions of the file are reserved as keys to locate information in other parts of the file. A list structure is one wherein each data element contains the address of, and thereby points to, a successor element. A ring structure is a circular list structure, in which the last data element points back to the first.

These file structures should not be confused with the access methods discussed earlier. The access methods relate to the set of routines a programmer may use to store and retrieve data records from secondary storage devices. The data management file structures refer to a type of file organization that permits a user to classify and store data. The data management routines will utilize whatever access methods are available to implement the various file structuring forms.

Secondary Storage Allocation - Techniques for allocating storage to data files vary widely from system to system. While the more sophisticated systems dynamically allocate and re-allocate storage as required, many systems leave this function almost entirely to the user. In the latter case, associated functions such as protection from inadvertent destruction may also be under user control. At either end of the spectrum, however, the function generally includes production of requested or periodic reports describing the status of allocated and unallocated secondary storage.

Input Transaction Processing - Input transaction processing provides for defining input data element formats, validating the data elements as they are entered, converting them into internal formats, and storing them in the data file. Input format definitions may be established prior to the entry of the data into the system or the data entry may be self-defining.

Prior to acceptance, the data elements may be subjected to one or more validation tests. Range verification may be used to insure that a data element is between a minimum and a maximum value, or to determine if an element is less than or greater than a specific value. An internal comparison test may be used to compare data elements to those already accepted by the system and stored within the data base. A masked comparison may be used to examine one or more specific characters within the data element. Sequence checking may be used to locate data elements which are not entering the system in accordance with an established ordering. The system may be designed so that an element which does not satisfy its validation criteria is rejected or is conditionally accepted and flagged in a description of the transaction processing. Additionally, some systems provide for operator intervention to accept, reject or correct data which does not satisfy the criteria.

Logical and Interactive Maintenance - Logical file maintenance permits conditional or programmed updating of a data file. Logical maintenance may or may not be transaction oriented. If it is, the transaction updates the object file only when specified criteria are satisfied. Non-transaction oriented maintenance is usually accomplished via internal actions generated by a computer pseudo-language program. For example, a logical maintenance job might specify the deletion of every record written after a given calendar date, or conversely, the retention of only those records written between two given calendar dates.

Interactive maintenance, as its name implies, is the updating of a data file from an on-line terminal. Almost all interactive maintenance applications utilize logical file maintenance features. Prior to initiating the actual transaction the terminal user must usually establish logical parameters to isolate records of interest.

File Reorganization - This function provides for reorganization of one or more existing data files into a new composite output file. Data fields may be added, deleted, or changed in size or type under the restructuring process. For example, a single fixed-length field might be respecified as a repeating variable length field. When the reorganization involves more than one input file, the resultant output file is composed of several merged, and possibly restructured, data fields from each of the input files. This file maintenance function, unlike the others, is usually invoked on a rather limited basis. Consequently, the function is often performed by a self-contained adjunct to the system, and often uses its own control specifications.

1.3.3 Data Qualification and Retrieval

Almost all data management systems are designed so that data qualification and retrieval may be accomplished in either a batch or interactive mode. In the batch mode, the data base may be interrogated by individually prepared logical queries or by prestored logical queries in which specific operands can be altered. In the interactive mode interrogation may be by a cue-response form of query, by a prompting query which "guides" the user through the interrogation, or by a user-programmed query. Each record satisfying the parameter of the query may be directly displayed, retained on an intermediate file for subsequent processing, or passed on to another portion of the data management system, such as data output or file maintenance. Thus, a single data qualification scheme generally serves the entire data management system.

The basic format of a qualification statement takes the form of a data base field, a logical operator, and an operand. For example, AGE EQUALS 14, where AGE is a field in the data base, EQUALS is the Boolean operator signifying equality, and 14, a numeric operand for comparison. While almost every system permits the full range of Boolean operators (equal, greater than, less than, not equal, etc.), some systems provide additional types such as quantitative or occurrence operators, arithmetic or statistical operators, and application-dependent operators. Complex statements may frequently be formed by joining simple qualification expressions with the logical connectors AND, OR, NOT and by nesting simple qualifications at several levels. The operand of a qualification statement may be a constant, another data field, the result of another qualification statement in the same retrieval, or an arithmetic expression.

Some data management systems limit qualifications and retrieval statements to a single file search. Others permit several files to be searched as the result of a single request. The more sophisticated systems also provide inter-file searching whereby a second file is searched for qualifying information based upon the results of the primary file search.

1.3.4 Data Output

Almost all data management systems produce certain standard reports such as a listing of directories or a listing of input transactions processed against a file. Additionally, data management systems often provide one or more methods of report definition. In some systems this may take the form of report program generators or the capability to include user programs developed by programming technicians. Additionally, many systems include schemes whereby an interactive or batch user with limited programming interests can influence the form of output data. In either case, the data output function of a data management system normally operates either on the results of a user's retrieval or on an entire system file. Reports may normally be produced on printed listings, punched cards, tape, or on-line display devices.

Controls of data output include labeling, data formatting, and pagination control. Labeling consists of specifying page headers, trailers and data labels. Data formatting includes character insertion or deletion, affixing mathematical or commercial symbols, punctuation, and element decoding prior to output. Pagination provides top-of-form control for printed output and forward/backward spacing frame control for CRT devices.

Both the capability to control these items and the extent of intrinsic support vary widely among systems. For example, some systems will include support facilities to develop horizontal and vertical displays appropriate to the indicated output device, while others leave such processes entirely to the user, and simply truncate if the output exceeds device capacity.

2.0 DATA HANDLING UTILITIES

The data handling utility programs are generally rather uncomplicated routines of general usefulness to the installation. Classically, these routines were independent programs which were loaded and executed when required. However, in contemporary systems they have been incorporated into the operating system and are activated by a program call, a system control card or an operator key-in. These utilities rarely interface directly with an executing program, though they are frequently included as a separate job step in a multi-step job.

2.1 Display Facilities

The utility routines that produce visual output as final products are termed display facilities. While the printer is the most common output medium, CRT's, console typewriters, etc., may also be utilized by various display routines.

The most common display facilities provided are for main storage (core dumps), system catalogs, tables and directories. Other display facilities are generally provided for data stored on any secondary storage media supported by the system. Typically, these include tape to printer, disk to printer, etc. Since these media conversions are normally performed expressly for visual display to human consumers, they often incorporate special formats and visual aids in addition to simple data translation (e.g., octal to decimal). Generally, the formats and visual aids are predetermined, although some systems allow the user to influence the output by exercising certain options.

2.2 Peripheral Device Support

The utility routines that provide peripheral device support perform such functions as volume positioning, media conversion, data editing, and test file support. Volume positioning may be used to backspace, forward space, or rewind a magnetic tape, or to locate a file on a tape or direct access volume, or to locate a specific record within a file.

The media conversion facilities simply permit data to be copied from one secondary storage medium to another. Generally, these facilities are provided for all combinations of peripheral devices supported by the system. Typical examples are: card to tape, tape to card, tape to disk, etc. Of course, these transfers between media may be accompanied by additional formatting or data translation. Unlike the display facilities discussed above however, these transfers are not performed expressly for human consumption. As a consequence, media conversion tends to be straightforward and occurs within the limits of the media involved. Thus, a card to tape conversion may be expected to duplicate the data from each card in sequential 80 character tape records, etc.

An extension of the media conversion facilities discussed above provides facilities for dumping and reloading secondary storage. These facilities are generally employed for backup file creation and for storage compaction.

Additional data handling features available in many systems support file, rather than device, oriented processing. These features are included here, however, due to their basic similarity to the media conversion facilities. For example, data editing facilities permit the user to scan a data file to detect logical data errors (e.g., out-of-sequence conditions) and to compare files residing on separate media. Furthermore, several systems provide test file support utilities which can be used to develop device-oriented data files for use in application program testing.

3.0 SORTING AND MERGING

Programs that sort or merge sets of data records normally comprise a part of the operating system. The sorting function takes strings of unordered records and sequences them according to a given key or set of keys within each record. The merging function, on the other hand, takes separate strings of records which have already been ordered by keys and produces a composite ordered output string. The design of most sort programs is such that the unordered records are placed into ordered strings and then merged. Consequently, a single program usually suffices to perform both functions.

3.1 Sort Module Development

The first phase of the sorting or merging process is to develop the programming module which will actually perform the sorting or merging function. Two basic approaches are used. The first provides a general set of tables which enable a general purpose sort program to tailor itself to the specific sorting requirements. The other actually generates a special purpose sort program specifically designed for the application.

In both cases, control parameters are interpreted that describe the characteristics of the available hardware: available core space and the amount and type of intermediate storage available (e.g., tape, disk, etc.). Additional control parameters describe the characteristics of the data: record size, number and relative location of control fields, number of records, etc. Finally, still other control parameters are introduced that permit special options: ascending or descending sequences, user-specified collating sequences, special data conversion requirements, etc. The control parameters are normally edited, and if no errors are found, they are used as the basis for developing the executable sort module.

3.2 Sort Module Execution

The sort module execution phase normally consists of three separate steps: input reading and string creation, intermediate string sequencing, and final string merging.

The first step reads the input data from wherever it is located, isolates the sort key, and modifies the input record by any user-written routines that have been specified. The purpose of the first step is to create a series of sequenced strings from the unsequenced input. Consequently, as input records are read, they are sorted internally and written onto an intermediate device as a sequenced string. Three types of sorting are possible: a record sort where the full record is written out; a field select sort, where certain fields are deleted from the record prior to writing it out; and a tag sort, where the record is stored on an intermediate device and only its sort key and address are used. Several internal sorting techniques are used to create the sequenced strings. Among the more popular are tournament, exchange, radix, and sieve sorts.

The second step merges the strings produced in the first step until there are as many strings as the order of merge. Thus, if the order of merge is four, then the strings would be merged until four or less sequenced strings remained. The types of merging techniques used in this phase vary with the characteristics of the intermediate devices available. For example, if magnetic tapes are used as the intermediate storage device and if the tape drive has a read backward feature, a polyphase merge is the most economical. Other techniques used are: balanced merges, cascade sorts, and oscillating sorts. Each technique has advantages and disadvantages based upon the hardware capabilities and the number of intermediate storage devices.

The final step is the merge proper operation whereby the strings are merged one more time to create a single sequenced string. If a merge function was initially specified, the first two steps are bypassed and only this step is executed. In either case, the records are reformatted to their final output appearance and written on the output device. Provision is usually made for user-written routines to modify each record prior to placing it on the output device. When a tag sort is used, the user may also be given the option of accepting the sorted record address list rather than producing the actual output string.

APPENDIX I

GLOSSARY OF OPERATING SYSTEM TERMINOLOGY

This Glossary is intended to provide the most commonly accepted definitions of operating system terminology. An Addendum to the Glossary is provided which discusses uncommon or conflicting definitions and terms. Relevant Addendum entries are referenced in the Glossary.

-A-

abnormal termination: Termination of a job or task due to an error condition (contrasted with normal termination).

address translator: A software or hardware feature which dynamically translates virtual instruction and data addresses to real main storage addresses. (See virtual address, real address.)

algorithm: A prescribed set of well defined rules or processes for the solution of a problem in a finite number of steps.

algorithmic dispatching: Dispatching according to a series of rules and decisions.

algorithmic scheduling: Scheduling according to a series of rules and decisions.

allocate: To grant a resource to, or reserve it for, a job or task.

alternate routing:

1. Access of an I/O device via an alternate channel when an I/O device is connected to the CPU by more than one channel.
2. Transmission of output to a secondary device when the primary device is inoperable.

application software: That software oriented to specific problem solution. (See also: problem program.)

arm (interrupt): To allow the occurrence of an interrupt (opposite of disarm). (See Addendum, interrupt.)

assembler: A language processor which converts symbolic instructions into a form suitable for execution on a computer.

asynchronous: Without regular time relationships (opposite of synchronous); hence, as applied to program execution, unpredictable with respect to time or instruction sequence.

attach:

1. To reserve a system resource for the exclusive use of a single program.
2. To cause the execution of a dependent subtask.

automatic restart: A restart which is initiated by the system supervisor independently of either user or operator direction.

-B-

background: A partition or group of partitions used for processing batch jobs. The background is normally assigned the lowest priority in a multiprogramming environment. (See also: foreground.) (See Addendum, background.)

batch processing: Processing of jobs which are submitted to run independently of events outside the system (as opposed to real time or interactive jobs) and are normally processed on a deferred or time-independent basis (e.g., whenever the processing work load is light). (See Addendum, batch processing.)

batch query: A query processed in the batch processing mode. (See query.)

baudot code: The standard five-channel teletypewriter code consisting of a start impulse and five character impulses, all of equal length, and a stop impulse whose length is 1.42 times all of the start impulse. This code is also known as the 1.42 unit code.

benchmark: A standard program used to evaluate the performance of computers relative to preselected criteria.

binding: Transforming one or more object modules into a composite program which is acceptable for execution. (Also called collecting (Univac), and linkage editing (IBM and RCA); See Addendum, linkage resolution).

block (records):

1. (v) To group records for the purpose of conserving storage space or increasing the efficiency of access or processing.
2. (n) A physical record so constituted or a portion of a telecommunications message defined to be a unit of data transmission.

blocking: The combining of two or more records into one block.

bootstrap: A technique for loading the first few instructions of a routine into storage; then using these instructions to bring in the rest of the routine.

breakpoint: A point in a computer program at which conditional interruption (to permit visual check, printing out, or other analyzing) may occur. Breakpoints are usually used in debugging operations.

buffer (input/output): A portion of storage into which data is read, or from which it is written. (See also: chained segment buffer.)

buffering: A method of managing I/O buffers (e.g., simple or exchange buffering, locate or move mode buffering).

buffer pool: A group of buffers which may be allocated as needed to various jobs.

byte: A generic term used to indicate a measurable portion of consecutive binary digits; e.g., an 8-bit or a 6-bit byte (most commonly, 8 bits). (See also: word.)

-C-

catalog:

1. (n) A directory to locations of files and libraries.
2. (v) To enter an item into a directory.

cataloged procedure: A set of job control statements which has been placed in a special file and which may be used for job control by being named on a special control card.

chained segment buffer: A buffer composed of a chain of fragmented core areas with each area containing a pointer to the next area in the chain. Normally used to process records of varying lengths such as teleprocessing messages.

channel: A hardware device which connects a CPU and main storage or an I/O processor with I/O control units or devices.

channel queue: A queue of requests for service from a data channel.

channel scheduler: The part of the supervisory program which controls the movement of data between main storage and input/output devices.

character string (record): An unformatted record composed of a series of contiguous characters; usually applied to teleprocessing messages. (See also: message.)

check: The occurrence of an error (e.g., machine check, program check). Usually causes an interrupt.

checkpoint: (See also: restart.)

1. A point at which information about a program is recorded which will permit subsequent resumption of processing from that point.
2. The recording of such information.

collector: (See linkage editor/loader.)

command processing: The reading, analyzing, and performing of commands submitted via the console device or an input job stream.

common core: An area in main storage shared by two or more programs or program segments.

common page: A page allocated to more than one program, usually used for passing data between separate programs.

common subroutine: A subroutine which may be used by more than one executing task; only one copy of the subroutine resides in main storage.

communication region: An area of the supervisor set aside for inter-program and intra-program communication.

compiler: A language processor which translates a problem-oriented language into a machine language.

compiler interfaces (operating system): Operating system functions that are oriented to providing supporting services to the system language compilers.

compressed format: A format for storing data in a minimum number of characters, generally achieved through the elimination of blanks.

concurrent: Existing or occurring during the same interval of time. (See also: simultaneous.)

concurrent processing: Processing two or more programs (or tasks) on an interleaved basis.

console: The interface, or communication device, between a user or operator and the computer (e.g., operator's console, remote terminal/console).

contention: Rivalry for use of a system resource.

control block: A storage area through which a particular type of information required for control of the operating system is communicated (e.g., message control block).

control message: A finite sequence of letters, digits, symbols, etc. transmitted to convey regulatory information.

control program: A collective or general term for all routines or programs which are part of the operating system supervisor.

control specification: The act of defining to a set of routines the functions that they are to perform.

conversational: Synonymous with interactive.

core dump: To transfer the contents of all or part of main storage to a peripheral device.

core image library: A library of load modules.

core resident: The condition of a program or table which is in main storage.

core resident routine: A routine which executes from and remains permanently in main storage (opposite of transient routine).

core storage: See main storage.

core storage access control: Preventing executing programs from retrieving and/or storing data in core storage areas not assigned to that program. (See also: storage protection.)

CPU (central processing unit): The portion of a computer which directs the sequence of operations, interprets the coded instructions, and initiates the proper commands to the computer circuits for execution. (See Addendum, CPU.)

cue-response query: A form of data management system interrogation in which the user participates in a question/answer dialogue with the system. (See query.)

-D-

data access control: Any of the data management techniques available to the user for transferring data between main storage and an I/O device; (e.g., direct, indexed, keyed, queued or sequential access).

data base:

1. generic: The entire collection of information available to a computing system.
2. specific: A structured collection of information as an entity or collection of related files treated as an entity.

data code translation: Translation of data from one symbolic representation to another; e.g., EBCDIC to USASCII-8.

data handling utilities: Utility programs which provide a variety of independent services to manipulate and/or display various groupings of data elements.

data management: Comprehensive facilities which provide support for programmed access to the data files within the system. These facilities may be of two forms: routines supporting application program access to the data base and the independent data management system supporting user access to the data base.

data management system: A group of integrated routines developed to create and maintain a large, organized and structured collection of related data (known as the data base) and to interrogate the data base and produce various types of formatted reports.

data manipulation functions: The components of the operating system that permit the user to access and process data. These functions may either be independent utility programs or subroutines incorporated within a user program.

data qualification: The process of isolating an element of a data file; usually for some purpose such as retrieval, file maintenance, etc.

data set: Synonymous with file.

data transcription: The process of converting data from one peripheral medium to another (e.g., card to tape or mass storage to printer).

data transfer: The movement of data either within main storage, between main storage and secondary storage, or between main storage and an input/output device.

deallocation: Restoring the availability of a system resource.

deblocking: Isolating the individual records within a block (See block, blocking).

dedicated memory: Core memory locations reserved by the system for special purposes, such as interrupts and real-time programs.

default option: An option which will automatically be assumed if not overridden by a parameter specification.

deferred restart: A restart which is initiated by operator or user action, and usually involves resubmitting the job.

dependent program: Any user program or non-supervisory system program.

device: A term used to refer to a computer component.

device category (I/O): A term used to collectively describe several different I/O devices having common processing characteristics, such as direct access devices, sequential devices, etc.

device manipulation (I/O): Control functions that allow a physical I/O device to be positioned without involving data transfer. These functions typically include: rewinding and/or unloading tapes, ejecting printer pages, stacking punched cards, etc.

device name: The general name for a device, specified at the time the system is generated, and used for all symbolic references to the device.

device queue: A queue of requests for processing service from an I/O device.

device resolution: Determination of a specific physical device from a symbolic device name.

device type (I/O): A term used to collectively describe several different I/O devices having common physical characteristics, such as tape devices, disk devices, etc.

diagnostic:

1. Pertaining to the detection and isolation of hardware or software malfunctions.
2. A message or record, recording the occurrence of a hardware or software error.

diagnostic error processing: The recognition of the occurrence of error conditions within the system along with the corresponding corrective actions.

diagnostic logout: The detailed record of hardware status, including hardware registers, switches, etc., taken at the occurrence of a hardware or software failure.

direct access:

1. A method of accessing data records directly, without regard to the sequence in which they are recorded. (See also: access control.)
2. A device oriented to support of this method of access, e.g., disk, drum, etc.

directory: See catalog (n).

disarm (interrupt): To disallow the occurrence of an interrupt (opposite of arm). (See Addendum, interrupt.)

dispatcher queue: A queue of tasks that are ready for dispatching.

dispatching: Allocation of processor time by the supervisor to a specific task. Tasks that are eligible for dispatching have already been placed in an execution state by the scheduler and are not waiting for I/O activity, operator responses, etc.

driver: A program which controls the use of a peripheral device.

dynamic allocation: Providing resources (storage space, I/O devices, etc.) to a program or task in response to an actual demand by the program or task.

dynamic buffering: A buffering technique which provides buffers to a program or task in response to an actual demand for buffers by the program or task.

dynamic dump: A dump which is performed during the execution of a program.

dynamic program loading: The process of loading a program module into main storage, upon the demand for that program module by an executing program.

dynamic program relocation: Moving or relocating a program to another part of storage without modifying it, before it has completed execution, and still permitting subsequent execution.

dynamic storage: Storage which is dynamically allocated to tasks during program execution. It is normally used for subroutines, buffer pools, etc.

dynamic system maintenance: The on-line capability whereby a system reconfigures itself:

- 1) to adjust to operator addition or deletion of some of its processing elements;
- 2) to fallback into a degraded state of operation upon the failure of some of its processing elements. (See also: system maintenance, system degradation, system reconfiguration.)

-E-

error recovery: Synonymous with recovery processing.

error severity code: A code indicating the severity of errors noted by an assembler or compiler and used to determine subsequent processing of the resulting object module.

event: An occurrence of significance to a task; typically, the completion of an asynchronous operation, such as input/output.

event monitoring: Maintaining control over executing programs. Such control includes processing all interrupts, trapping all error conditions, notifying programs of external condition requiring attention, and, in a multiprogram environment, allocating CPU time among contending programs.

event synchronization: Delaying task execution until some specified event occurs or triggering a task upon the occurrence of a specified event.

exchange buffering: A buffering technique which eliminates the need to move data in main storage by exchanging a system buffer area for a user buffer area.

execute: To carry out an instruction or run a program on the computer.

executing state: The state of a program during the time it is using the CPU.

executive/control functions: The components of the operating system that maintain real-time execution control over the system environment.

executive routine support (compiler interface): Special facilities which may be provided by the executive to the compiler that are not available to other problem-oriented programs, e.g., the use of communication tables within the resident executive. (See compiler interfaces).

-F-

fault analysis: The analysis of hardware or software malfunctions by the system for purposes of error recovery (including system reconfiguration).

file: A collection of related records treated as a unit. The records in a file may or may not be sequenced according to a key contained in each record.

file access control: Preventing unauthorized access to a file. The protection may extend to read access, write access, or both.

file location recognition: The identification of a permanent data file via its system-assigned or user-assigned label.

file maintenance (interactive): The facility that provides for interactive updating of a data file from an on-line terminal.

file maintenance (logical): The facility that permits the conditional or programmed updating of a data file.

file management routines: The collection of operating system routines that are used to accomplish all the various file services provided by the system, such as file control, cataloging, file protection, etc.

file purging: The act of destroying the contents of a file. The name of the file may or may not be removed from the system catalog.

file reorganization: The facility that supports the restructuring of one or more existing files.

file security control: Restricting access to a file or device to only those users who have been authorized to use it.

firmware: Software and hardware which interact so closely that the functions of both become intertwined.

fixed length record: A record having the same length as all other records in the same file. (Contrasted with variable length record.)

fixed logic query: A query in which the operands and operators cannot be altered by the user at execution time (as opposed to skeletal or interactive queries). (See query.)

foreground: The partition or partitions which are normally used for real time, communications, and time sharing applications. The foreground is usually given priority over the background in a multiprogramming environment. (See also: background.) (See Addendum, background.)

foreground processing: Processing of programs in the foreground partitions.

foreground scheduler: A routine which analyzes processing requests and executes the appropriate foreground programs.

format: The arrangement of data.

free core pools: Areas of main storage which may be dynamically allocated as I/O buffers or program expansion areas.

-G-

generated code: The executable programs or subroutines produced from a set of specifications or commands.

-H-

hardware: Physical equipment, e.g., mechanical, electrical, or electronic devices (contrasted with software.)

header label: The record at the beginning of a file or volume containing control information about the file or volume. (See also: trailer label.)

hierarchical file structure: A file structure in which the elements are classified and stored according to a ranking scheme.

I/O control: The initiation, execution, monitoring and control of data transfers into and out of the system.

index:

1. A table in the catalog structure used to locate files.
2. A table used to locate the records of an indexed file.

indexed access: An access method in which record locations are determined from an index or table.

indexed file structure: A file structure in which one or more fields within the file are indexed. In addition to storing the individual records, indexed files maintain a directory of the values of indexed fields and the corresponding locations of all records containing each value.

initialize: To set various counters, switches and addresses to prescribed starting values, at the beginning of, or at specified points in a computer routine.

input: Data transferred or to be transferred from an external storage medium into the internal storage of the computer.

input queue: The input data to a job or to the system.

input reader: A program which reads the input job stream and writes the information onto mass storage where it is kept until resources are available for execution of the program.

input (job) stream: The sequence of control statements and data submitted to the operating system on an input unit especially designated for this purpose at system generation time, or by the operator.

input transaction processing: A term which refers to the updating of designated records via a set of specific transactions. The updating may consist of the addition, deletion, or modification of one or more fields within each record. Each transaction contains the identification of the record it is to update.

inquiry: A technique whereby interrogation of a computer system or program may be initiated at a keyboard.

interactive: A conversational mode of processing which permits interaction or dialogue between a user and a time-sharing system (contrasted with non-interactive).

interactive query: A query formulated and posed on-line to a data management system.
(See query.)

interface:

1. The place at which two different systems (or subsystems) meet and interact with each other.
2. The linkages and conventions established for communication between two independent elements; usually between a program and another program, computer operator, terminal user, etc.

interfile search: A multi-file search in which the answers obtained from the results of querying one file are used to interrogate a second file. (See multi-file search.)

interleave: To arrange parts of one sequence of things or events so that they alternate with parts of one or more other sequences of things or events and so that each sequence retains its identity.

interrupt: (See also: am, disam, mask, unmask.) (See Addendum, interrupt.)

1. (n) A break in the normal flow of a system or routines such that the flow can be resumed from that point at a later time. An interrupt is usually caused by a hardware-generated signal.
2. (v) To cause an interrupt.

interrupt stacking: Recording the occurrence of one or more interrupts and allowing them to remain pending while processing continues.

interval timer: A counter which is automatically incremented or decremented at regular time intervals and which normally causes an interrupt when reaching zero. (See also: real-time clock.)

-J-

job: A total processing application comprised of one or more related processing programs, such as a weekly payroll, a day's business transactions, or the reduction of a collection of test data. (See Addendum, job.)

job class: A parameter which, in some systems, permits a measure of control over the sequence of program execution.

job control: A general term which collectively describes those functions of the control program that regulate the use, by jobs, of the central processing unit and other resources except input/output devices.

job control language: The language used to provide job specifications to the job control routines.

job management: The real-time initiation, scheduling, monitoring and control of normal system operations together with the necessary allocation of resources.

job priority: A designation given by the user to each of his jobs to specify the relative sequence in which the jobs are to be executed.

job processing: The reading of control statements from an input stream, the initiating of job steps defined in these statements, and the writing of output messages.

job queue: The stack of jobs to be executed by an operating system.

job scheduler: The control program function which controls input job streams, obtains input/output resources for jobs and job steps, and otherwise regulates the use of the computing system by jobs.

job status: The status of a job at a given point in time, e.g., position within a job queue or state of activity.

job step: That unit of work associated with one processing program and related data. (See Addendum, job.)

-K-

key:

1. A data item which serves to uniquely identify a data record.
2. A code used by a program to gain access to protected main storage areas, I/O files, or I/O devices. (See also: lock, password.)

keyed access:

1. An access method in which record locations are determined algorithmically from keys.
2. An access method dependent upon hardware recognition of physically recorded keys.

keyed file structure: A file structure in which a data item is used to uniquely identify each data record. The search for a specific record by using the record key is implemented by hardware on some systems, by software on others.

key-in: Information entered by an operator via a keyboard.

label processing: Those facilities provided for writing and checking both standard and non-standard labels when a data file is opened or closed.

language processor: A compiler, assembler or program generator which transforms source modules into object modules.

library: In general, a collection of objects (e.g., files, volumes, card decks) associated with a particular use, and the location of which is identified in a directory of some type (e.g., core image, private, public or system libraries.)

library and directory maintenance: The capabilities provided to create and update different types of libraries and directories.

library support (compiler interface): Support provided to maintain compiler-oriented libraries. These libraries may be compiler source program libraries, macro statement libraries, or compiler subroutine libraries. (See compiler interfaces).

linkage: The means by which communication is effected between two routines or modules.

linkage editor/loader: A program which produces a load module by transforming object modules into a format acceptable for execution and resolving all inter-module linkages. In some systems the resulting load module is located in main storage and may be immediately executed; in others, the load modules must be subsequently loaded for execution. (See Addendum, linkage resolution.)

list file structure: A file structure wherein each data element points to a successor element.

load: To read a load module into main storage preparatory to executing it.

loader: A program which takes load modules and places them in main storage.

load module: A program in a format suitable for loading into main storage for execution. (See also: module, object module, source module.)

load module generation: The process of converting machine language program instructions into a format capable of being loaded directly into main storage and executed.

locate mode: A buffering technique in which data is pointed to rather than moved.

lock: A code used by an operating system to provide protection against improper use of main storage areas, I/O files or I/O devices.

lockout: A programming technique used to prevent access to system resources when these resources are currently in use.

logical I/O: Conceptual I/O operations which are performed by user programs through use of the system and which may, in fact, involve no physical I/O but only blocking or deblocking (contrasted with physical I/O.)

logical record: A record which is defined in terms of its use without regard to the way in which it is recorded.

logout:

1. (v) To record the occurrence of an event.
2. (n) The documentation of the occurrence of an event.

-M-

machine check: An error due to hardware malfunction; usually causes an interrupt.
(See check.)

main storage: A digital computer's principal working storage from which instructions can be executed or operands fetched for data manipulation. Also frequently referred to as memory or core storage (contrasted with secondary storage.)

management support utilities: Utility programs provided for the use of the system manager. These programs perform a wide variety of functions in support of initializing, testing, and monitoring the system.

manual re-configuration: System re-configuration performed through key-ins or control cards.

mapping device: A hardware address translator.

mask (interrupt): To suspend recognition of an interrupt (opposite of unmask). (See Addendum, interrupt.)

mass storage: Secondary storage capable of holding large amounts of data; generally, a direct access device.

memory:

1. Synonymous with main storage.
2. A device in which data can be stored and from which it can be retrieved.

memory dump:

1. The transfer of the contents of main storage to a secondary storage device.
2. A print-out (generally edited or formatted) resulting from a memory dump.

message: A quantity of transmitted information that is physically contiguous and processed as a unit.

message block: A set of characters transmitted as a unit to or from a remote terminal in a communications environment. A message block is usually variable in length.

message control: A function of the operating system which controls the transfer of data between programs and remote terminals.

message control block: Information generated for each incoming or outgoing message in a communications environment. The MCB governs the transfer and processing of the message. Included in the MCB is the identification of the remote terminal and the address of each block of the message to be processed. (See also: control block.)

message queue: A list of communication messages.

message routing: Interpreting a code internal to a teleprocessing message and, on input, directing the message to the attention of the proper program, and on output, directing the message to the correct output device.

message time stamping: Appending the time of message receipt to the message.

module (programming): The input to, or output from, a single execution of a language, processor, or linkage editor; a source, object, or load module; hence, a program unit that is discrete and identifiable with respect to compiling and loading. (See also: relocatable module.)

move mode: A buffering technique by which data is moved between the buffer and the user's work area.

multi-file search: A data retrieval which interrogates several files.

multiprocessing: The employment of multiple interconnected processing units to execute two or more different programs or tasks simultaneously. (See Addendum, multiprocessing.)

multiprogramming: A general term that expresses use of the computing system to execute two or more different programs or tasks concurrently. (See Addendum, multiprogramming.)

-N-

non-interactive: Processing during which there is no dialogue between the user and the system (contrasted with interactive).

normal termination: Termination of a job or a task upon successful completion of processing (contrasted with abnormal termination).

-O-

object module: The output of a single execution of a language processor, which either constitutes input to a linkage editor, or is a load module. (See also: module, source module.)

off-line: A generic reference to a device or operation which is not directly controlled by the computing system with which it is associated.

on-line: A generic reference to a device or operation which is directly controlled by the computing system with which it is associated.

on-line diagnostics: Diagnostic messages which are output to an operator or user console.

operating system: A set of programs and routines which guide a computer in the performance of its tasks and assist the programs (and programmers) with certain supporting functions.

operating system management: The generation and maintenance of the computer operating system.

operator: The individual who monitors the computer system and performs necessary physical actions, such as mounting tapes, setting switches, etc., as required by the system.

operator command: A statement to the control program, issued via a console device, which causes the control program to provide requested information, alter normal operations, initiate new operations, or terminate existing operations.

operator's console: A console used by the computer operator, generally containing special manual controls not available on a user's console.

output: Data transferred or to be transferred from the internal storage of the computer to an external device or onto a permanent recording medium (paper, cards, etc.)

output stream: Diagnostic messages and other output data issued by the operating system or the processing program to output units especially designated for this purpose at system generation time or by the operator.

output work queue: A queue of control information describing system output files, which specifies to an output writer the location and disposition of system output.

output writer: A symbiont which transcribes specified output files onto an output unit, independently of the program which produced such files.

overlap: To do something at the same time that something else is being done; for example, to perform input/output operations while instructions are being executed by the central processing unit.

overlay program: A segmented program in which the segment currently being executed may use the same core storage area occupied by a previously executed segment.

overlay segment: The smallest functional unit which can be loaded as one logical entity during execution of an overlay program.

owner: The creator of a file, who may control access to the file by other users. (See file access control.)

-P-

page: A program section of a convenient size for transmission between main storage and secondary storage.

paging: A technique of loading sections (pages) that are referenced during program execution.

parameter: A definable characteristic of an item, device, or system, which can be used to control or modify an algorithmic process such as a subroutine, a system generation program, etc.

parity check:

1. (n) A machine check which occurs due to the failure of a word or byte to contain an even number of bits (even parity) or an odd number of bits (odd parity).
2. (v) To test a word or byte for proper parity.

partial dump: To transfer part of main storage to a peripheral device. (See also: storage dump.)

partition: A subdivision of main storage which is allocated to a job or a system task for job or task execution. A partition may be fixed or variable in size. (See Addendum, partition.)

password: A word used to allow the user access to protected storage, files, or I/O devices. (Synonymous with key, definition 2; related to file access control, storage and volume protection.)

patch:

1. (n) A section of coding inserted into a routine to correct a mistake or alter the routine. It is often not inserted into the actual sequence of the routine being corrected, but placed somewhere else, with an exit to the patch and a return to the routine provided.
2. (v) To insert such corrected coding.

periodic interrupts: Interrupts scheduled to occur at a periodic interval.

peripheral:

1. A generic term referring to devices that are not a part of the computer main frame (e.g., card readers, printers, tape units, etc.).
2. Referring to an operation or device which is off-line.

physical I/O: I/O which takes place between main storage and on-line devices (contrasted with logical I/O).

physical record: A record defined in terms of the way it is recorded without regard to its use.

polling: A technique by which each of the terminals sharing a communications line is periodically interrogated to determine if it requires servicing. (See teleprocessing.)

pool: A collection of interchangeable peripheral devices or core storage locations. When the user requests a device or area from the pool, the system selects an available device or area to associate with the job (e.g., buffer, or free core pool).

post-mortem dump: A memory or file dump performed upon completion of program execution, frequently associated with abnormal termination.

pre-stored query: A query which has been stored in a system library. At execution time it may be loaded directly from the library. (See query.)

priority: An order of precedence established for competing events.

priority queue: A queue maintained in priority sequence.

priority scheduling: A form of job scheduling which schedules jobs by priority rather than by their sequence in the input job stream.

private library: A library which is accessible only by a restricted group of users.

privileged instruction: An instruction which can only be executed in supervisor mode.
(See Addendum, privileged instruction.)

problem mode: A hardware or software supported mode of operation in which certain privileged instructions cannot be executed by a program, (contrasted with supervisor mode). (See Addendum, privileged instruction.)

problem program: Any of the class of routines or programs which perform processing of the type for which a computing system is intended, including routines that solve problems, monitor and control industrial processes, sort and merge records, perform computations, process transactions against stored records, etc. (similar to user program).

processing support: The routines within the supervisor which accomplish a variety of miscellaneous services for an application program.

processor:

1. From a hardware point of view, a device which performs one or many functions; usually means a central processing unit (CPU). (See Addendum, CPU.)
2. From a software point of view, a processor is a program, usually an assembler or compiler, which transforms some input into some output.

program: The complete sequence of instructions and routines necessary to solve a problem (e.g., control, problem, service or user program, or a symbiont).

program check: An error that occurs due to a programming error such as an invalid instruction, invalid address, etc.

program expansion area pools: Areas of main storage which may be dynamically allocated to programs.

program generator: A language processor which combines and tailors general purpose object modules based on source control statements.

program library: A collection of available computer programs and routines.

program limit monitoring: Assuring that an executing program or task does not exceed certain system or user-specified limits. Limits are typically established for: CPU time, main storage space, output cards/lines, etc.

program loading: The process of placing load modules in main storage. (See also: loader.)

program maintenance: The support functions provided to the user for the maintenance, and modification of application programs.

program termination processing: The processing functions that are executed when a program or task comes to either a normal or abnormal termination.

prompting query: An aid provided by some data management systems which assists a user in formulating meaningful retrieval statements by "leading" him through the interrogation process. (See query.)

public library: A library available to all users of a system.

push-down list: A list which is constructed and maintained so that the next item to be retrieved is the most recently stored item in the list, i.e., last-in, first-out.

-Q-

quantum: A limited or algorithmically specified interval of time.

query: A statement specifying the retrieval criteria necessary to locate specific information in a file. (See batch query, cue-response query, fixed-logic query, interactive query, pre-stored query, prompting query, and skeletal query.)

queue: A list of entries which identify things waiting for service or attention.

queue control: The system functions required to add elements to, remove elements from, and update elements within system queues.

queued access: An access method in which records are read prior to being requested, and queued in main storage, thus eliminating a wait for transmission from an I/O device when a record is requested.

-R-

random access:

1. (v) A method of accessing data records without regard to the sequence in which they are recorded.
2. (n) A device oriented to support this method of access.

read only: An attribute of programs or files whereby they are not modified during execution of a program.

ready state: The condition of a task which is in contention with other tasks for use of the central processing unit and is awaiting assignment of the CPU. All requirements for its activation have been satisfied.

real address: An instruction address which refers to an actual location in main storage.
(Contrasted with virtual address.)

real time: (See Addendum, real time.)

1. Pertaining to the actual time during which a physical process transpires.
2. Pertaining to the performance of a computation during the actual time that the related physical process transpired in order that results of the computation can be used in guiding the physical process.

real-time clock: A program accessible clock which indicates the passage of actual time. The clock may be updated by hardware or software. (See also: interval timer.)

record: A collection of related items of data, treated as a unit (e.g., fixed length, logical, physical, undefined, or variable length records).

recovery processing: Action performed by a user or system program in response to the occurrence of an error. (See also: restart.)

recursive: Repetitive on a cyclical basis; a procedure which, while being executed, either calls itself or calls another procedure which in turn calls the original one.

reenterable: Variation of reentrant.

reentrant: The attribute of a load module which allows the same copy of the load module to be used concurrently by two or more tasks. The attribute is attained by programming the load module in such a manner that it does not modify itself. (See also: reusable.)

refreshable: A refreshable module can be replaced by a new copy during execution by a recovery management routine without changing either the sequence or the results of the processing. Thus, it is a module that is never modified by itself or any other module during execution.

relocatable module: An object or load module which can be relocated in core.

relocation:

1. Loading a program into main storage at an address other than that specified at assembly or compilation time.
2. Moving a program module from one area of main storage to another.
3. The modification of a program module required to effect relocation as defined above.

remote batch processing: The submission of jobs for batch processing from a remote terminal. (See also: batch processing.)

remote terminal/console: An interface or communication device between a user and a computer, generally located away from the computer installation. (See also: console.)

request stacking: Placing a request for processing (system service, I/O , etc.) in a queue which will be serviced at some later point in time.

resident supervisor: That portion of the supervisor which remains in main storage at all times.

resource: Any facility of the computing system or operating system required by a job or task. These include main storage, input/output devices, the central processing unit, files, and control and processing programs.

resource allocation: Assigning a system resource for the use of a partition, job, job step, or task.

resource deallocation: Removing a system resource from a partition, job, job step, or task.

resource status modification: Altering the availability or definition of system resources. Typically, I/O devices may be added or deleted from the operating environment, partition sizes modified, etc.

restart: To re-establish the status of a system or job at some previous point in time (usually performed as a part of recovery processing). (See also: automatic restart, checkpoint, deferred restart.)

retrieval (data): The act of locating and selecting specific information in a file.

return code: A code which is established by a program or subroutine to notify the system or calling program of its terminal status.

reusable: The attribute of a routine that permits the same copy of the routine to be used by two or more tasks. (See reentrant, serially reusable.)

ring file structure: A circular list structure in which the last data element points back to the first. (See list structure.)

roll-out/roll-in: A method of increasing available main storage by temporarily storing an executing program on secondary storage, using the area occupied by the program, and then returning the program from secondary to main storage and restoring its status.

root segment: That segment of an overlay program which remains in main storage at all times during execution of the overlay program.

round-robin scheduling: A technique for allocating CPU time to a number of contending programs (a type of time-slicing). The technique involves establishing a circular list of users and allocating a fixed amount of time to each user in turn without regard to priority.

routine: A set of instructions arranged in proper sequence to cause the computer to perform a desired task; usually part of, or executed as part of, a program (e.g., a subroutine).

-S-

scheduler: A system component whose function is to allocate all resources for a job and to perform all initialization operations necessary prior to job execution.

scheduling: That system function which prepares a job for execution. (See also: dispatching.)

scheduling algorithm: The rules and decisions by which a program is scheduled.

scheduling queue: A queue of jobs that are ready for scheduling.

secondary storage: The storage facilities not an integral part of the computer but directly connected to and controlled by the computer; e.g., magnetic drum and magnetic tapes (contrasted with main storage).

segment:

1. The smallest functional unit of an overlay program which can be loaded as one logical entity.
2. As applied to telecommunications, a portion of a message which can be contained in a buffer of a specified size.

selective trace: A tracing routine wherein only instructions satisfying certain specified criteria are subject to tracing. Typical criteria are: (a) instruction type; (b) instruction location; (c) data location. For case (a), where tracing is performed on transfer or branch instructions, the term logical trace is sometimes used.

sequential access:

1. A method of accessing data such that consecutive records are processed sequentially. (See also: access control.)
2. An I/O device oriented toward the above method of access, e.g., tape drive, card reader, etc.

sequential file structure: A file structure in which elements are stored serially.

serially reusable: The attribute of a routine such that, when in main storage, the same copy of the routine can be used by another task after the current use has been concluded. (See: reusable.)

serial processing: Processing which occurs on a sequential basis, i.e., execution of a task or program only upon completion of the preceding task or program.

service program: Any of the class of standard routines or programs which assist in the use of a computing system and in the successful execution of problem programs, without contributing directly to control of the system or production of results, and including utilities, simulators, test and debugging routines, etc.

session: The period of time during which the user engages in a dialogue with a time-sharing system.

shared device: A device which may be concurrently used by two or more users.

shared file: A file which may be accessed by two or more users.

shared routine: A routine which can be concurrently executed by several users.

simple buffering: A buffering technique which assigns buffers permanently to a program until the program is terminated.

simple program structure: A program structure consisting of one module which occupies a fixed amount of main storage.

simulation:

1. Use of a computer system to represent or model some other system, e.g., another computer system, a traffic situation, etc.
2. (I/O) A supervisory function which handles a problem program's I/O request without performing the actual data transfer. Used almost exclusively for testing/debugging.
3. (system) Routines which will simulate various operating system conditions or functions. These routines may be used to test and validate the operating system or application programs running under operating system control.

simultaneous: Existing or occurring at the same instant of time. (See also: concurrent.)

skeletal query: A pre-stored query which is written in outline or skeletal form. The user may define the specific operators, operands, etc. at execution time. (See also: query, pre-stored query.)

slave: An element of a computing system that is under the functional control of a similar element.

slave computer: A computer that is under the functional control of another computer.

slaved terminal: A terminal that functions as an I/O device for another terminal rather than as an independent terminal.

snapshot (dump): A memory dump performed during execution of a program; generally used as a debugging aid.

software: The collection of programs and routines associated with the computer (e.g., application or system software; contrasted with hardware).

sorting and merging: Those routines that sequence strings of records according to keys within each record.

sorting technique: A scheme by which a set of data items are rearranged into a new sequence of records or strings.

sort module: The programming module that actually performs a sorting or merging function.

source module: A series of statements in the symbolic language of a language processor which constitutes the entire input to a single execution of the processor. (See also: load module, module, object module.)

stacked job processing: A technique of automatic job-to-job transition with little or no operator intervention. (See also: batch processing.) (See Addendum, batch processing.)

stand-alone utilities: Routines which are not under operating system control during execution.

standard option: A system option for which a default was established during system generation. (See also: default option.)

status display: The visible presentation of the current state of the system or of some component of the system.

storage compacting: A procedure for maximizing available main storage area by re-locating active programs from fragmented to contiguous areas.

storage device: A device into which data can be entered, in which it can be held and from which it can be retrieved.

storage dump: To transfer the contents of all or part of main storage to a peripheral device.

storage protection: Protection of an area of main storage against unauthorized access (Read, write or both).

string: A connected sequence of elements, e.g., a bit string or a character string.

structure control: The capability of managing different program structures during the program loading process. Some representative program structures are: simple, overlay, and dynamic.

subroutine: A routine which is called by another routine to perform a specific function.

subtask: A task which is created by and is subordinate to another task.

supervisor: The programs which schedule, allocate, and control system resources rather than process data.

supervisor call (monitor call, master mode entry, SVC): A request by a program or task for a service to be performed by the supervisor.

supervisor mode: A hardware or software-supported mode of operation in which all operations may be performed, (contrasted with problem mode). (See Addendum, privileged instruction.)

surface analysis: A reliability check conducted to detect failing surface area. Normally, surface analysis is conducted for direct access devices, though it may also be performed on magnetic tape.

suspended: The state of a program whose execution has been halted pending the occurrence of some specified event.

swapping: A method of sharing main storage between several programs by maintaining each program and its status on secondary storage and loading each one into main storage for a limited time interval. (See also: time-sharing.)

symbiont: A data transfer routine or program which executes concurrently with user and system programs.

symbolic I/O: Reference to an I/O device by a symbolic name.

synchronous: Occurring concurrently, and with a regular or predictable time relationship (opposite of asynchronous).

system: An assembly of software and hardware integrated to form an organized whole.

system communication: Information exchange between the user or operator and the operating system.

system definition: Creation of tables which determine the way in which general-purpose system programs will function; may be part of system generation.

system degradation: System reconfiguration (normally due to a malfunctioning unit) which results in poorer system performance.

system description maintenance: The operating system functions performed to maintain updated tables of the current state of the system. These tables may represent static conditions, such as the parameters specified at system generation, or dynamic conditions, such as the number of current system users.

system generation: A process which creates a particular and uniquely specified operating system. System generation combines user-specified options and parameters with manufacturer-supplied general-purpose or non-specialized program subsections to produce an operating system (or other complex software) of the desired form and capacity.

system initialization: The process of loading the operating system into the computer and defining the processing environment.

system input device: An I/O unit specified as a source of an input job stream.

system interrogation:

1. Operator-initiated communication requesting system status.
2. Problem program-initiated interrogation of the supervisor requesting system status.

system library: A library available to all users of a system.

system maintenance: The process of updating the operating system in response to changes in the operating environment or changes and modifications to programs within the operating system itself. (See dynamic system maintenance).

system management functions: The non-real-time components of the operating system which support and maintain both system and application programs.

system measurement routines: Those routines which enable the system manager to obtain various statistics about the operational use of the system. The system manager may use these statistics to better adapt the operating configuration to the actual processing environment.

system output device: An I/O unit to which system output is directed.

system reconfiguration: Modification of an operating system to recognize a change in system resources.

system residence device: The external storage device allocated for storing the basic operating system.

system service request: A request by an executing task for the operating system to perform a service function. Typical service functions are providing I/O services, timer services, scheduling services, storage dumps, etc.

system software: Software developed as a general purpose tool to supplement the computer hardware. Systems software normally include operating systems, compilers, assemblers, etc., but does not include application or problem oriented programs. (Also referred to as basic software.)

system startup: The process of loading and initializing an operating system.

system status: The status of an operating system at a given point in time, e.g., number of current jobs, status of system resources, etc.

system test mode: A distinct operating environment in which the executing program is being tested. Systems having this feature provide special debugging facilities which are not available to programs executing in production mode.

system utility program support (compiler interface): Linkage sequence generation capabilities provided by some compilers to users of the language so that they can reference various system utility programs. (See compiler interfaces.)

-T-

task: A program subdivision which is treated as the basic unit of work by the supervisor. (See also: subtask.) (See Addendum, job.)

task suspension: Temporarily suspending the execution of a task while continuing to keep the task under system control. A suspended task is removed from the dispatching queue but is not treated as a terminated task.

teleprocessing: A general term expressing data transmission between a computing system and remotely located devices via a unit which performs the necessary format conversion and controls the rate of transmission. (See also: polling.)

terminal: See remote terminal/console.

third generation: A term used to characterize the general-purpose digital computers introduced in the late 1960's. The term is generally applied to three separate computer characteristics: electronic hardware components, logical organization, and software or programming techniques. (See Jordain, Condensed Computer Encyclopedia, pp. 525-529 for an excellent discussion of this term.)

third generation hardware: Hardware constructed from integrated circuits.

third generation logical organization: Special features that provide the ability to handle many programs at the same time. Such features include: memory-protection circuits, hardware address modification, modular components (CPU's, memory, data channels), and telecommunication capabilities.

third generation software: Software supporting an operating system that is indispensable to the normal functioning of the computer. Functions are divided between hardware and software logic so as to maximize joint efficiency.

time-sharing: (See Addendum, time-sharing.)

1. The simultaneous utilization of a computer system from multiple terminals.
2. Performing several independent processes almost simultaneously by interleaving the operations of the processes on a single processor.

time slicing: The allocation of limited intervals of time (quantums) to programs in contention for use of the CPU (e.g., round-robin scheduling).

timing service: Service functions provided by the operating system to an executing program that utilize one of the system timers. Typical functions include: providing the real clock time, providing notification of an elapsed interval time, suspending the executing program for a specified interval, etc.

trace: An interpretive diagnostic technique which provides an analysis of each executed instruction and writes it on an output device as each instruction is executed.

track replacement: The substitution of an alternate track for a track determined to be defective during surface analysis. (See surface analysis).

trailer label: A record placed at the end of a file or volume which contains control information about the file or volume. (See also: header label.)

transient area: A main storage area usually, but not necessarily, within the supervisor area used for executing transient routines.

transient routines: Routines permanently stored on the system resident device and loaded into a transient area when needed for execution. Generally, they accomplish selected supervisory functions but are not executed often enough to merit inclusion in the resident supervisor.

turnaround time: The elapsed time between submission of a job to a computing center and the return of results.

-U-

undefined record: A record having a length unspecified or unknown to the system.

unmask (interrupt): To allow recognition of an interrupt (opposite of mask). (See Addendum, interrupt.)

user: Anyone who requires the services of a computing system.

user account: An account maintained within the system for each system user. This account typically contains user identification information as well as accumulated charges and statistics of computer system utilization.

user program: A program written by a user to solve a specific problem, i.e., not a supervisory or control program (similar to problem program).

-V-

variable length record: A record having a length independent of the length of other records in the same file (contrasted with fixed length record).

virtual address: An instruction address which refers to a relative or imaginary location in main storage and which must be algorithmically converted to a real address before the instruction is executed (contrasted with real address).

virtual storage: A conceptual extension of main storage achieved via a software or hardware technique which permits storage address references beyond the physical limitations of main storage. Virtual addresses are equated to real addresses during actual program execution.

volume: All that portion of a single unit of storage media which is accessible to a single read/write mechanism, e.g., a reel of tape, removable disk pack, etc.

volume maintenance: Those functions which provide diagnostic information for, correct error conditions on, and remove unwanted elements from any volume in the system. (See surface analysis and file purging.)

volume preparation: The functions invoked when a new volume is to be added as a system component. These functions include the writing of standard volume labels, formatting records and/or tracks, creating volume directory entries, etc.

volume protection: Protection of a volume against unauthorized access (read, write or both).

-W-

waiting state:

1. The state of a program which is idle pending the occurrence of some event, e.g., completion of an I/O operation.
2. The state of a program on secondary storage which is waiting to be swapped into core for execution, e.g., a program in a time-sharing system.

word: A generic term used to indicate a measurable portion of consecutive binary digits; usually the equivalent of two or more characters or bytes.

GLOSSARY

ADDENDUM

This Addendum is a supplement to the Glossary providing additional information about terms defined in the Glossary. It addresses three subjects: (1) conflicting definitions from various sources; (2) information which is pertinent to, but does not define terms presented in the Glossary; and (3) collective discussion of interrelated terms.

Presenting these subjects as an Addendum rather than including them in the main body of the Glossary serves two purposes. First, the Glossary maintains its primary purpose, to provide commonly accepted definitions of OS terminology, and does not confront the reader with masses of text unnecessary to the definition of those terms. Second, the Addendum permits a freer format, more suited to the discussion of several terms as a group or the presentation of tables showing, for instance, variations in the terminology used by various manufacturers.

The following terms are discussed in the Addendum:

- background, foreground
- batch processing, stacked job processing
- CPU (central processing unit), processor
- interrupt (arm/disarm, enable/disable, mask/
unmask, trap)
- job (job step, task)
- linkage resolution
- multiprocessing, multiprogramming
- partition (region, set)
- privileged instruction (problem mode, supervisor mode)
- real time
- time-sharing

background, foreground: (See also: batch processing, real time, time-sharing.)

These terms apply to functionally oriented areas in main storage in a multiprogramming system. The background area is normally used for batch or stacked job processing while the foreground is used for real-time and time-sharing jobs, although in some systems time-sharing may also be considered a background operation. In general, high priority programs are foreground programs and low priority programs are background programs. CDC, in fact, in its MSOS documentation eliminates the definitions of background and foreground and simply defines priorities. The terms background and foreground may also be used to refer to the types of programs run in these areas. Other definitions are listed below:

- background: "on a time-sharing system, tasks that are executed non-conversationally; i.e., the batch processing workload." ... IBM, 360 TSS.
- background program: A "running program, " i.e., not an interrupt servicing routine ... Digital, PDP-8.
"in multiprogramming, the background program is the program with lowest priority. Background programs execute from a stacked job input." ... IBM, 360 DOS/TOS.
- background area: "that area of core storage allocated to batch processing." ... XDS, Sigma 5/7 BPM.
- background processing: "...is the normal, stacked-job processing of the job stream in a communication or data transcription environment." ... Honeywell, Series 200 Mod 2 (Extended) OS.
- foreground: "A generic reference to tasks being executed by a time-sharing system, while there is dialogue or conversational interaction between the user and the system." ... IBM, 360 TSS.
- foreground processing: "Processing of real-time programs in the memory area immediately following the resident control routines." ... Honeywell, Series 200, Mod 2 (Extended) OS.
- foreground program: "The interrupt service routine." ... Digital, PDP-8.
"In multiprogramming, foreground programs are the highest priority programs." ... IBM, 360 DOS/TOS.
"a real-time program executed on the occurrence of a priority interrupt signal or an unsolicited key-in." ... XDS, Sigma 5/7 BPM.

batch processing, stacked job processing: (See also: real time, time-sharing.)

Both terms are normally used to refer to processing jobs from a job stream as opposed to real-time processing or time-sharing.

Historically, the technical definition of batch job processing referred to a technique that used a single program loading to process many individual jobs (such as a single compiler load to process many compilations). Honeywell distinguishes batch from stacked processing as follows:

"Stacked-job processing is a refinement of the batched-job approach. In batched-job processing, a single processing function, e.g., compilation, is applied to all jobs in the batch. In stacked-job processing, any number of processing functions, such as compilation, maintenance, and execution, can be successively applied to the same job. Thus, each job in the input stack is processed to completion before the next job is accepted."

However, the distinction between these two terms is no longer common and batch job processing has become synonymous with stacked job processing - a technique of automatic job transition with little or no operator intervention.

Other definitions of these terms are listed below:

- batch: "...an object program running in a stacked job manner. Shares the central processing unit with the priority program when a priority program is present and executes only when the priority program is not in control of the processor." ... CDC, 31/32/33/3500 MSOS.
- batch processing: "a technique by which items to be processed must be coded and collected into groups prior to processing." ... Datamation ADP Glossary.
"A mode of operation where several runs are grouped prior to processing. Transition from run to run is effected by the executive system." ... UNIVAC, 1108 EXEC 8.
- stacked job processing: "A technique that permits multiple job definitions to be grouped (stacked) for presentation to the system, which automatically recognizes the jobs, one after another. More advanced systems allow job definitions to be added to the group (stack) at any time and from any source, and also honor priorities." ... IBM, 360 OS.

CPU (central processing unit), processor:

CPU is probably the most widely used term to describe the primary control unit of a computing system. The term processor (or variants, e.g., processor module, central processor) is also widely used; however, "processor" may also be used to refer to a variety of control units in a modular system. For instance, some common types of processors are central processor modules (program control and operations units), memory modules (main storage and memory control), and input/output control (IOC) modules (controlling data transfer between memory modules and I/O devices). The GE 600 system is an example of a modular system with several types of processors.

Listed below are terms used to describe the unit (or units, in a multiprocessing system) containing the major control circuitry in a variety of computing systems.

- central processor: Burroughs (B8500), CDC (31/32/33/3500), GE (615/635), Honeywell (Series 200)
- computer: CDC (1700)
- CPU (central processing unit): Digital (PDP-8), IBM (360), XDS (Sigma 2, Sigma 7), SEL (Systems 86)
- mainframe section (of computer): IBM (1800)
- processor: Burroughs (B2500/B3500, B6500), HP (2116B), Honeywell (Model 8200), RCA (Spectra 70)
- processor module: Burroughs (B8500), GE-615/635

interrupt (arm/disarm, enable/disable, mask/unmask, trap):

Systems which provide an interrupt feature provide the capability of masking interrupts (to defer recognition to them) and unmasking interrupts (to allow immediate recognition of them). The XDS Sigma 2 and 7 systems have this capability, but refer to masking and unmasking as disabling and enabling, respectively. The Sigma 2 and 7 also provide the additional capability of preventing any recognition of interrupts. Disarmed interrupts will be totally ignored; armed interrupts will be held pending if they are masked or recognized if they are unmasked. The word trap is frequently used synonymously with interrupt; it may also be used to refer to status information trapped (saved) when an interrupt occurs.

job (job step, task):

Work submitted to an operating system in a job stream can be classified at one of three levels. At the highest level is a job encompassing all the programs required for the execution of a given application. In extended multiprogramming systems, several jobs may be scheduled to execute concurrently; however, the individual programs within each job are processed serially.

At the intermediate level is the job step, an individual program to be scheduled for loading and execution. Job steps are specified by separate execute cards within the job.

At the lowest level is a task which is the smallest independently executable block of coding. This block may be a separate program or part of a program. The execution of a task is directed by an internal command from another executing program. The following table shows the terms used in various systems to describe these levels. Note that all three levels are not defined in all operating systems.

	<u>Job</u>	<u>Job Step</u>	<u>Task</u>
Burroughs (MCP)	-	Job/Program	-
CDC:			
MASTER	Job	Task	Task
MSOS	Job	Run	-
Digital (Multiprogramming and Swapping Monitors)	-	Job/Program	-
GE (GECOS-III)	Job	Activity	-
Honeywell:			
Mod 1 (TR) OS	Job	Program	-
Mod 2 (Extended) OS	Job	Program Step	-
Mod 4 OS	Job	Job Step	-
IBM:			
DOS/TOS	Job	Program	-
OS	Job	Job Step	Task
SDC (ADEPT)	Job	Program	-
SEL (810A/810B OS)	-	Job/Program	-
RCA (POS, DOS, TDOS)	Job	Program	-
UNIVAC:			
EXEC-8	Run	Task	Activity
494 OS	Job	Task	Activity

linkage resolution: In most systems, capabilities exist to combine the object modules produced by various language compilers with subroutines and other object modules to produce a single program or task. In order for this to occur, the subroutines and object modules (with the possible exception of the base module) must be relocatable with respect to assigned storage addresses. When these modules are combined, the assigned storage addresses of the relocatable modules are modified so that no two modules are assigned the same core area; instead the modules are normally assigned to contiguous storage locations. Once these locations are determined, the intermodule linkages (whereby one object module transfers control and data to another) are resolved by determining the newly assigned storage locations of the respective modules and inserting these addresses into the linkage sequence.

Many systems also offer an additional capability that will permit an automatic search of a system library for any object modules or subroutines with a name the same as that in an unresolved linkage sequence. These systems will then load and relocate these additional modules.

The program that performs this function occurs in two different forms depending upon the system. In some systems, it is an independent job step that must be scheduled explicitly; in others, it is a supervisor service that may be dynamically invoked by an executing task. Furthermore, the resulting load module may, in some systems, be immediately executed, while in others it may require subsequent loading as part of a separate job step.

The names assigned to this process and to the program that performs the process are:

	<u>Process</u>	<u>Program Name</u>
Burroughs	binding	binder
CDC	linking	loader, relocatable loader
DEC:		
PDP-9 Keyboard Monitor	linking	linking loader
PDP-10 Multiprogramming Monitor	linking	relocatable loader
GE	linking	loader
Honeywell	linking	linkage loader
IBM:		
S/360 1800	linkage editing building	linkage editor builder
RCA	linkage editing	linkage editor
XDS:		
Sigma 2 BCM	linking	linking (relocatable) loader
Sigma 5/7 BPM	linking	loader
SEL	linking	relocatable loader
UNIVAC:		
1108 EXEC-8	collecting	collector
418-III OS	collection phase	job loading
494 Omega OS	collection	loader
9400 OS	collecting	linkage editor

multiprocessing, multiprogramming:

The Glossary defines multiprocessing and multiprogramming as simultaneous and concurrent execution of programs, respectively. "Multiprocessing" is almost always used to mean simultaneous execution of programs by multiple processing units. "Multiprogramming" is normally used to mean interleaved execution of programs, but may be used in a more general sense to refer to any method of concurrently executing programs, including multiprocessing.

The USA Standard Vocabulary for Information Processing (1966), on the other hand, defines multiprocessing as concurrent and multiprogramming as interleaved execution. Thus according to USASI, multiprocessing subsumes multiprogramming. However, most sources agree with the definition presented in the glossary.

Several definitions of multiprocessing and multiprogramming follow:

- multiprocessing:

"The use of two or more computers to logically or functionally divide jobs or processes and to simultaneously execute various programs or segments of programs asynchronously." ... CDC, 33/3500 MASTER

"Two or more processors simultaneously executing programs in memory to gain greater throughput." ... GE, GECOS-III

"Two or more processes running simultaneously in one computer system." ... Honeywell

"The simultaneous use of two or more processing units in the same computing system." ... IBM, 360 TSS

"...employing two or more interconnected processing units to execute programs simultaneously." ... IBM, 360 OS

- multiprogramming:

"...a technique for processing two programs simultaneously by overlapping or interleaving their execution." ... CDC, 31/32/33/3500 MSOS

"The concurrent processing of many programs residing in core memory to maintain the highest possible amount of simultaneous input/output, and to maximize processor utilization." ... GE, GECOS-III

"Two or more programs operating simultaneously in one computer system." ... Honeywell

"A processing technique whereby more than one program coexists in computer memory, simultaneously contending for system resources." ... Honeywell, Series 200 MOD 2 (Extended) OS

"A general term that expresses use of the computing system to fulfill two or more different requirements concurrently." ... IBM, 360 OS

"A technique by which a computing system can be used to execute two or more unrelated programs, parts of which reside in main storage." ... IBM, 360 TSS

"The concurrent execution of several programs which occupy main storage. This is accomplished by sharing the attentions of the central processor." ... UNIVAC, 1108, EXEC 8

partition (region, set):

In most multiprogramming systems, the user area is divided into partitions; each partition can be used for the execution of one program (a time-sharing controller and the programs swapped in and out of an area are usually treated as one program by the multiprogramming supervisor). These partitions may be fixed size (determined at system generation time or by the operator) or variable size (expanded or contracted dynamically by the system in response to calls from the problem program).

The Honeywell Mod 4 system also defines a set which is a fixed area of core containing one fixed partition or two variable partitions. IBM OS documentation uses the term partition to mean fixed size partition (MFT) and defines region as a variable size partition (MVT).

privileged instruction (problem mode, supervisor mode):

Many current computing systems provide two operating modes: a problem mode in which certain privileged instructions cannot be executed and, a supervisor mode in which all instructions can be executed. The purpose of these modes is to prevent the problem programmer's mistakes from disrupting system operation by disallowing his use of instructions vital to the integrity of the system. I/O, interrupt handling, and storage protection instructions are typically privileged.

Computing systems which include this feature and the terms used to designate each state are listed below. Note that not all systems having this feature have terms associated with the states.

	<u>Supervisor Mode</u>	<u>Problem Mode</u>
Burroughs (B2500/B3500)	control state	normal state
CDC (3300/3500)	monitor state	program state
GE (600 Systems)	master mode	slave mode
Honeywell (Series 200)	---	standard processing mode
IBM (360)	supervisor state	problem state
RCS (Spectra 70)	privileged mode	non-privileged mode
XDS:		
Sigma 2	---	---
Sigma 7	master mode	slave mode
SEL (810B)	privileged state	unprivileged state
UNIVAC (1106/1108)	---	guard mode

real time: (See also: batch processing, time-sharing)

Several definitions are listed below.

- real time:

"Pertaining to a program for which time requirements are particularly stringent; that is, the data processing must keep up with a physical process within a time period of seconds or less." ...CDC, 33/3500 MASTER

"Pertaining to computation performed while the related physical process is taking place so that results of the computation can be used in guiding the physical process." ... Digital, PDP-8.

"(1) Pertaining to the actual time during which a physical process transpires. (2) Pertaining to the performance of a computation during the actual time that the related physical process transpired in order that results of the computation can be used in guiding the physical process." ... Honeywell

"The actual time during which a physical process transpires, especially if that process is monitored or controlled by a computing system." ... IBM, 360 TSS.

- real-time processing:

"An environment in which communications data is received from remote terminals, processed, and returned to these terminals within a reasonable time frame. Each user appears to have sole possession of the system while, in fact, system resources are time-shared among the users demanding service." ... Honeywell, Series 200 Mod 2 (Extended) OS. (Note: This definition corresponds with the glossary definition of time-sharing rather than real time.)

"An operating environment in which the response to external stimuli is sufficiently fast to achieve a desired objective. Depending upon the application, the response time may vary from seconds to microseconds. Generally, real-time processing is under the influence of asynchronous inputs from one or more devices." ... UNIVAC, 1108, EXEC 8.

time-sharing: (See also: multiprocessing, real time)

Time-sharing usually implies real-time use of a computing system by several users, concurrently. Unlike multiprogramming, where several programs reside in main storage simultaneously, time-sharing is generally accomplished by swapping programs into main storage from a high-speed secondary storage device. Several definitions of time-sharing are provided below.

- time-sharing:

"the use of a device for two or more purposes during the same overall time interval, accomplished by interspersing component actions in time." ... Datamation ADP Glossary.

"The capability of a computing system to accommodate more than one user during the same interval of time without apparent restriction by the existence of other users. In time-sharing, a given device is used in rapid succession by a number of other devices or various units of a system are used by different users or programs." ... CDC, 33/3500 MASTER

"a method of allocating central processor time and other computer services to multiple users so that the computer, in effect, processes a number of programs simultaneously." ... Digital, PDP-8 Family

"the interleaving of the time of a device." ... Digital, PDP-8/1.

"Using a device, such as a computer, to work on two or more tasks, alternating the work from one task to another. Thus, the total operating time available is divided among several tasks, using the full capability of the device." ... Honeywell

"A method of using a computing system that allows a number of users to execute programs concurrently and to interact with them during execution." ... IBM, 360 TSS.

"Pertaining to the interleaved use of the time of a device." ... UNIVAC, USASI

APPENDIX II

BIBLIOGRAPHY OF

OPERATING SYSTEMS DOCUMENTATION

BURROUGHS CORPORATION (BURROUGHS)

<u>Computer</u>	<u>Document Title</u>	<u>Form Number</u>
B2500/B3500	Master Control Program Operating Instructions	1040730 3-69
B2500/B3500	Master Control Programs Information Manual	1031218 5-69
B2500/B3500	Systems, Characteristics Manual	1025517 3-69
B300/B5500	Data Communications Information Manual	1030400 4-67
B5500	Information Processing Systems, Master Control Program Reference Manual	1042462 6-69
B5500	Time Sharing System, Terminal User's Guide	1038205 1-69
B6500	Master Control Program Information Manual	1042447 6-69
B6500	System Information Manual (Copy #162)	Preliminary Copy
B8500	Introduction to the B8500 Information Processing System	Brochure

CONTROL DATA CORPORATION (CDC)

<u>Computer</u>	<u>Document Title</u>	<u>Form Number</u>
1700	Computer Reference Manual	60153100D
1700	Input/Output Drivers, Software User's Manual	84760700B
1700	Mass Storage Operating System 2.0 Operating Guide	60191400A
31/32/3300	SCOPE Utility Reference Manual	60130200
31/32/3300	AUTOLOAD Utility Reference Manual	60130100

31/32/33/3500	LISA/MASTER/MSOS, Reference Manual	60236900
31/32/33/3500	MSIO/MSOS, Reference Manual	60191300
31/32/33/3500	MSOS, Reference Manual	60173000B
31/32/33/3500	Real-Time SCOPE, Reference Manual	60172500A
31/32/33/3500	Data Processing Package, General Information Manual	60055100B
31/32/33/3500	Mass Storage Sort, General Information Manual	60172600
31/32/33/3500	MSOS, Installation Handbook	60237300
3300/3500	MASTER, Reference Manual	60213600C
3300/3500	MASTER, Installation Handbook	60214000
3600	Library Preparation and Maintenance	60083500
3600	SCOPE, General Information Manual	60132200
3600	Drum SCOPE, Reference Manual	60059200A
3600	SCOPE, Reference Manual	60053300B
3600/3800	Library Functions	60056400
64/65/6600	SCOPE 3, Reference Manual	60189400H
64/65/6600	SCOPE 3, Installation Handbook	60235600
64/65/6600	SCOPE Operator's Guide	60179600
6400/6600	SORT/MERGE General Information Manual	60173900

DIGITAL EQUIPMENT CORPORATION (DEC)

<u>Computer</u>	<u>Document Title</u>	<u>Form Number</u>
PDP-8	Introduction to Programming	C-18 1000 12/68
PDP-8/1	System User's Guide	DEC-08-NGCB-D
PDP-8/1	Timeshared-8	Brochure
PDP-8/1	Time-Sharing System, TSS/8 Monitor	DEC-T8-MRFA-D
PDP-9&15	Advanced Software System Monitors	DEC-9A-MADO-D

PDP-10	Single User Monitor System, Programmer's Reference Manual	DEC-10-MKDO-D
PDP-10	Timesharing Monitors: Multiprogramming Monitor (10/40)/Swapping Monitor (10/50), Programmer's Reference Manual	DEC-10-MTEO-D

GENERAL ELECTRIC CORPORATION (GE)

<u>Computer</u>	<u>Document Title</u>	<u>Form Number</u>
615/635	Information Systems Manual	CPB-371E
625/635	Comprehensive Operating Supervisor (GECOS-III) Reference Manual	CPB-1518A
	GECOS-III, Startup, Software Maintenance Document	CPB-1489A
625/635	Typewriter Messages (GECOS-III), Reference Manual	CPB-1477A
625/635	File and Record Control, Reference Manual	CPB-1003F
625/635	GECOS-III, I/O Supervision, Software Maintenance	CPB-1494
625/635	GERTS, Programming Reference Manual (GECOS-III)	CPB-1558
625/635	GECOS-III, Time Sharing System, Software Maintenance Document	CPB-1501
625/635	Programming Reference Manual	CPB-1004F
600 Line	GECOS-III, File System Maintenance	CPB-1497B
600 Line	Utility	CPB-1422C
600 Line	General Loader	CPB-1008F
600 Line	Bulk Media Conversion	CPB-1096F
600 Line	GECOS-III, Startup, Software Maintenance Document	CPB-1489A

HEWLETT-PACKARD COMPANY (HP)

<u>Computer</u>	<u>Document Title</u>	<u>Form Number</u>
HP 2116 B	Time Shared Basic System	Brochure
HP 2116 B	HP 2000A Time Shared Basic System, External Reference Specifications	Interim Specification

HONEYWELL, INC. (HONEYWELL)

<u>Computer</u>	<u>Document Title</u>	<u>Form Number</u>
Series 200	Summary Description	2-012
Series 200	Series 200 Programmers' Reference Manual, Models 200/1200/1250/2200/4200	2-139
Series 200	MOD 1 (TR) Operating System, Summary Description	3-258
Series 200	MOD 1 (MSR) Operating System, Summary Description	2-615
Series 200	MOD 1 (MSR) Mass Storage System Generation Instructions	3-866
Series 200	MOD 1 (MSR) Utility Routines	3-619
Series 200	MOD 2 (Extended) Operation System, Summary Description	5-393
Series 200	MOD 4 Operating System Summary Description	2-273
Series 200	Operating System - MOD 4, Systems Manual	0-A03
Series 200	MOD 4 Utility Programs	0-A05
Series 200	Operating System - MOD 8, Systems Manual	0-895

INTERNATIONAL BUSINESS MACHINES (IBM) CORPORATION

<u>Computer</u>	<u>Document Title</u>	<u>Form Number</u>
1800	Multiprogramming Executive Operating System, Programmer's Guide	C26-3720-1
System/360	System Summary	A22-6810-9
System/360	Basic Programming Support, Basic Operating System, Tape Operating System and Disk Operating System, Programming Systems Summary	C24-3420-1
System/360	Disk and Tape Operating Systems, Concepts and Facilities	C28-5030-5
System/360	Disk Operating System, System Control and System Service Programs	C24-5036-4
System/360	Disk Operating System, Supervisor and Input/Output Macros	C24-5037-2
System/360	Disk Operating System, Autotest Specifications	C24-5062-1
System/360	Disk Operating System, Tape and Disk Sort/Merge Program	C28-6676-2
System/360	Disk Operating System, Disk Sort/Merge Program	C24-3444-4
System/360	Disk Operating System, Tape Sort/Merge Program	C24-3438-4
System/360	Disk Operating System, System Generation and Maintenance	C24-5033-9
System/360	Disk Operating System, Utility Programs Specifications	C24-3465-5
System/360	Operating System, Introduction	C28-6534-1
System/360	Operating System, Concepts and Facilities	C28-6535-4
System/360	Operating System, System Programmer's Guide	C28-6550-4
System/360	Operating System, Messages, Completion Codes and Storage Dumps	C28-6631-1
System/360	Operating System, Programmer's Guide to Debugging	C28-6670-0
System/360	Operating System, Planning for Multiprogramming with a Fixed Number of Tasks (MFT)	C27-6939-3
System/360	Operating System, Job Control Language	C28-6539-7

System/360	Operating System, Time Sharing Option, Planning for TSO	C28-6698-0
System/360	Operating System, Job Control Language Charts	C28-6632-4
System/360	Operating System, Supervisor and Data Management Service	C28-6646-2
System/360	Operating System, Supervisor and Data Management Macro-Instructions	C28-6647-0
System/360	Operating System, Utilities	C28-6586-10
System/360	Operating System, Linkage Editor and Loader	C28-6538-8
System/360	Operating System, Sort/Merge	C28-6543-5
System/360	Operating System, Online Test Executive Program	C28-6650-2
System/360	Time Sharing System, Concepts and Facilities	C28-2003-3
System/360	Time Sharing System, System Generation & Maintenance	C28-2010-4
System/360	Time Sharing System, Time Sharing Support System	C28-2006-0
System/360	Time Sharing System, Terminal User's Guide	C28-2017-3
System/360	Time Sharing System, Independent Utilities	C28-2038-1
System/360	Time Sharing System, Linkage Editor	C28-2005-3
System/360	Time Sharing System, Command System User's Guide	C28-2001-5

RCA CORPORATION (RCA)

<u>Computer</u>	<u>Document Title</u>	<u>Form Number</u>
Spectra 70	All Systems Information Manual	70-00-501
Spectra 70	70/60 Processor Systems Information Manual	70-60-501
Spectra 70	Disc Operating System (DOS), Control System Reference Manual	70-00-619
Spectra 70	Primary Operating System (POS), Control System Reference Manual	70-00-603

Spectra 70	Tape Operating System (TOS), Control System Reference Manual	70-00-609
Spectra 70	Tape Operating System (TOS) Utility Routines	70-35-302
Spectra 70	Tape Operating System/Tape-Disc Operating System (TOS/TDOS) Programming System Information Manual	70-00-503
Spectra 70	Tape-Disc Operating System (TDOS), Control System Reference Manual	70-00-611
Spectra 70	Tape-Disc Operating System (TDOS), Sort/Merge System Program Publication	70-35-303
Spectra 70	Tape-Disc Operating System (TDOS) Utility Routines	70-35-306
Spectra 70	Time-Sharing Operating System (TSOS), Software Description Information Manual	79-00-516
Spectra 70	Time-Sharing Operation System (TSOS), File Editor Reference Manual	70-00-623
Spectra 70	Time-Sharing Operation System (TSOS), System Generation Reference Manual	70-00-622
Spectra 70	Time-Sharing Operation System (TSOS), Data Management System Reference Manual	70-00-614

SYSTEMS DEVELOPMENT CORPORATION (SDC)

<u>Computer</u>		<u>Form Number</u>
360/50	Time-Sharing System User's Guide: Overview	TM 3881/003/00
	Time-Sharing System User's Guide: User/Executive Interface & Summary of Executive Functions	TM 3881/007/01
	The Adept 50 System: A General Description of the Time-Sharing Executive and the Programmer's Package	TM 3899/100/00
	The ADEPT-50 Time-Sharing System	SP-3344
	Security Controls in the ADEPT-50 Time-Sharing System	SP-3342

360/50

Time-Shared Data Management System:
Technical Manuals

Professional Papers

TM 3849/000-
TM 3849/009
SP-2747, SP-2750,
SP-2907, SP-2634

SYSTEMS ENGINEERING LABORATORIES (SEL)

<u>Computer</u>	<u>Document Title</u>	<u>Form Number</u>
810B	SEL 810B General Purpose Computer (Brochure)	317-009101-003
810B	Reference Manual, SEL 810B General Purpose Computer	301-095118-100
810A/810B	Reference Manual, SEL 810A/810B Operating System	321-095071-000
810B	Real Time Executive Programming System (Brochure)	317-000001-001
Systems 86	Brochure	317-003600-000

UNIVAC DIVISION, SPERRY RAND CORPORATION (UNIVAC)

<u>Computer</u>	<u>Document Title</u>	<u>Form Number</u>
418-II	Real Time System, General Reference	UP-3929 Rev. 1
418-III	Real Time Operating System, Programmers Reference	UP-7690
494	Operating System, Programmers Reference	UP-7504 Rev. 1
1106	System Description	UP-7685
1107/1108	UNIMS Information Management System, System Applications Manual	UP-7674
1108	Operating System EXEC 8, Programmers Reference	UP-4144 Rev. 1

1108	Multiprocessor System, System Description	UP-4046 Rev. 1A
9200	System Description	UP-4086
9300	System Description	UP-4119
9300	Operating System, Programmers Reference	UP-7531 Rev. 1
9400	System Description	UP-7566
9400	Job Control for Disk Systems, Programmers Reference	UP-7585

XEROX DATA SYSTEMS (XDS)

<u>Computer</u>	<u>Document Title</u>	<u>Form Number</u>
Sigma 2	Sigma 2 Computer, Reference Manual	900964D
Sigma 2	Basic Control Monitor, Reference Manual	901064A
Sigma 2	Real-Time Batch Monitor, Reference Manual	901037A
Sigma 2	Real-Time Batch Monitor Subsystems, Reference Manual	901156A
Sigma 7	Sigma 7 Computer, Reference Manual	900950F
Sigma 5/7	Batch Processing Monitor, Reference Manual	900954B
Sigma 5/7	BTM: The Balanced System (Brochure)	647817A
Sigma 5/7	Batch Time-Sharing Monitor (BTM), Reference Manual	901577A
Sigma 5/7	Sort/Merge Reference Manual	901199C
Sigma 5/7	MANAGE Reference Manual	901610A

DOCUMENT CONTROL DATA - R & D

(Security classification of title, body of abstract and indexing annotation must be entered when the overall report is classified)

1. ORIGINATING ACTIVITY (Corporate author)

The COMTRE Corporation
151 Sevilla Avenue
Coral Gables, Florida 33134

2a. REPORT SECURITY CLASSIFICATION

UNCLASSIFIED

2b. GROUP

N/A

3. REPORT TITLE

ANALYSIS OF MAJOR COMPUTER OPERATING SYSTEMS

4. DESCRIPTIVE NOTES (Type of report and inclusive dates)

None

5. AUTHOR(S) (First name, middle initial, last name)

None

6. REPORT DATE

August 1970

7a. TOTAL NO. OF PAGES

123

7b. NO. OF REFS

1 plus Bibliography

8a. CONTRACT OR GRANT NO.

F19628-70-C-0258

9a. ORIGINATOR'S REPORT NUMBER(S)

ESD-TR-70-377

b. PROJECT NO.

6917

9b. OTHER REPORT NO(S) (Any other numbers that may be assigned this report)

10. DISTRIBUTION STATEMENT

This document has been approved for public release and sales; its distribution is unlimited.

11. SUPPLEMENTARY NOTES

12. SPONSORING MILITARY ACTIVITY

Deputy for Command and Management Systems
Hq Electronic Systems Division (AFSC)
L G Hanscom Field, Bedford, Mass. 01730

13. ABSTRACT

This report presents the results of an analysis of approximately 50 contemporary computer operating systems. The operating systems were analyzed for the purpose of defining an integrated functional classification structure applicable and common to the executive/control functions, system management functions and data manipulation functions of commercial operating systems. In addition to the functional capabilities of contemporary operating systems in terms of common identified functions, and selected implementation variations that occur within real-time, time-sharing, and batch processing environments. A glossary of operating system terminology is also included.

14.	KEY WORDS	LINK A		LINK B		LINK C	
		ROLE	WT	ROLE	WT	ROLE	WT
	operating systems (computer) functional analysis executive/control system management data manipulation data management multiprogramming batch processing real-time time-sharing						